5-2016

Implementation of the Continuous Space Language Model on a Heterogeneous Mobile Processor

Kurt Spencer Shively
Indiana University - Purdue University Fort Wayne

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Kurt Spencer Shively

Entitled

IMPLEMENTATION OF THE CONTINUOUS SPACE LANGUAGE MODEL ON A HETEROGENEOUS MOBILE PROCESSOR

For the degree of Master of Science in Engineering

Is approved by the final examining committee:

Elizabeth Thompson
Chair

Guoping Wang

Todor Cooklev

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Elizabeth Thompson

Approved by: Abdullah Eroglu 4/21/2016

Head of the Departmental Graduate Program Date

IMPLEMENTATION OF THE CONTINUOUS SPACE LANGUAGE MODEL ON A HETEROGENEOUS MOBILE PROCESSOR

A Thesis

Submitted to the Faculty

of

Purdue University

by

Kurt Spencer Shively

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Engineering

May 2016

Purdue University

Fort Wayne, Indiana

To my wife, for all of her understanding and support.

And to my mother, for instilling in me the value of an education.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Thomopson, for all of her time and effort on this research effort. Without her contributions, this research possible would not have been possible.

TABLE OF CONTENTS

Page

Page

Page

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

ABSTRACT

Shively, Kurt Spencer. M.S.E., Purdue University, May 2016. Implementation of the Continuous Space Language Model on a Heterogeneous Mobile Processor. Major Professor: Elizabeth Thompson.

Mobile processors continue to increase in performance while becoming more power efficient, providing consumers with improved gaming, multi-media, and browsing, along with longer lasting device usage. To keep up with consumer multimedia demand, mobile processor manufacturers have begun to integrate Graphical Processing Units (GPU) on mobile processors, providing users with the high performance graphics required for mobile gaming applications. The addition of integrated GPUs to the mobile processors also offers new opportunities for introducing to the mobile platform computationally intensive algorithms that were formerly impractical when running on the mobile CPU processor alone. GPU manufacturers such as NVIDIA are scaling down their GPU architecture from desktop systems to those developed for use on mobile processors, such as the Tegra K1, featuring a single GPU processing core.

This research effort investigates the performance of the computationally intensive Continuous Space Language Model (CSLM) algorithm on the NVIDIA Tegra K1 mobile processor and compares its performance on this platform to that of the desktop GPU platform reported by Thompson et al. [1]. The performance on the embedded GPU platform will also be compared to that of a conventional embedded CPU. However, first, the execution time of the algorithm will be observed while executing on a laptop CPU platform to provide a reference point for the performance of the Tegra K1 CPU processor(s). Next, the algorithm will be configured to execute solely on the Tegra K1 CPU processor(s), and the execution time will be observed. Finally, the execution time of

the algorithm will be observed after porting only the computationally intensive portions of the algorithm to the Tegra K1 GPU while other portions remain on the embedded CPU.

1. INTRODUCTION

This work involves porting a laptop CPU processor based version of Schwenk's [2] CSLM training algorithm to a mobile processor platform containing an integrated GPU and comparing the performance of the algorithm on the two platforms. In addition, various combinations of CPU-GPU hybrid operations are investigated, including offloading the computationally intensive portions of the CSLM training algorithm onto the embedded GPU, and leaving remaining operations to be performed on the CPU. Previous research by Thompson et al. [1] has shown that the performance of the CSLM training algorithm on desktop platforms using GPU cards significantly reduced the execution time of the training algorithm over that of a desktop CPU. The architecture of the mobile processor platform differs significantly from that of the desktop based GPUs used in Thompson et al. [1] in that the CPU processors and GPU are on the same chip and share the available on-board memory. Additional significant differences in the architecture of these two GPU platforms are described in detail in Sections 3 and 4. Given these differences, the execution time of the training algorithm using the GPU on the Tegra K1 is not expected to be less than that of the desktop GPU version, but a similar increase in performance over that of an embedded CPU is expected.

The method for this research will be similar to that followed by Thompson et al. [1] in that execution time of the embedded CPU version of the training algorithm will be used to compare to the execution time after porting the computationally intensive functions to the embedded GPU. The result of this work will yield a performance comparison between a desktop GPU and an embedded GPU implementation of the same computationally intensive neural network training algorithm.

2. THE CONTINUOUS SPACE LANGUAGE MODEL

2.1 Overview

Language models play a vital part in character and speech recognition and in translation and retrieval applications. Typical Statistical Machine Translation (SMT) systems consist of one or more translation models and a language model representing the target language. The goal of Statistical Machine Translation is to select, given a source sentence *f*, a target sentence *e* from all possible target sentences based on the maximal probability [2]. To do this, a probability model must be chosen to represent the relationship of the target sentence and the source sentence. A Bayes relationship, which is often referred to as the noisy source-channel approach shown in Equation 2.1, is the common method used in SMT to represent the probabilistic relationship between the target sentence and the source sentence [2].

$$\hat{e} = \underset{e}{argmax} Pr(e|f) = \underset{e}{argmax} Pr(f|e)Pr(e) \tag{2.1}$$

In Equation 2.1, $Pr(f|e)$ is the translation model from, for example, French to English, and $Pr(e)$ is the target language model (e.g., English) [2]. Traditionally, the language model used in statistical machine translation systems is the simple 3-gram or 4-gram back-off language model, used to generate *n*-best lists. Probabilities are used in language models to determine the next word in a multi-word sequence (referred to as *n*-grams) to train the model. The *n*-1 sequence of words is used as input to the model, and the output is the next word (*n*th) in the sequence based on the probability that it is the correct word. The performance of the 4-gram back-off language model has made it the standard approach used in SMT to generate the *n*-best lists used in scoring, where the first 3 words of the sequence are used to determine the 4th word of the sequence. This method does not allow “true interpolation” of the probabilities of the unseen *n*-grams since any

change in the discrete word space will result in an arbitrary change to the *n*-gram probability [3].

Schwenk's [3] Continuous Space Language Model provides an alternative to the *n*-gram back-off language model by using a neural network to first project the word indices onto a continuous space and then using a probability estimator over this space. Better generalizations of the unseen *n*-grams (that is, they are not in the training data but appear in the document to be translated) are observed, given the resulting word representations are smooth functions. Schwenk's language model implements an *n*-gram, but takes advantage of the ability of the neural network to learn the projection of the words onto the continuous space to estimate the *n*-gram probabilities. Given that the probability and estimation occur in a continuous space, the algorithm can perform meaningful interpolations with limited training data [3].

2.2 Probability Model

The goal of statistical language models is to be able to accurately predict the next word,w_t considering the complete word history w_1^{t-1} by learning the joint probability of the sequence of words in the language [4]. Equation 2.2 provides a statistical language model for all possible word strings w_1^T, based on the conditional probability of the next word given all previous words [4].

$$\hat{P}(w_1^T) = P(w_1) \prod_{t=2}^{T} \hat{P}\,(w_t | w_1^{t-1}) \tag{2.2}$$

In Equation 2.2, $\hat{\mathrm{P}}(w_1^T)$ denotes the probability of a given word sequence, $P(w_t|w_1^t - 1)$ is the probability of the next word in a sequence given the complete word history, and Π denotes the product of a sequence. Thus, the statistical language model defined in Equation 2.2 assigns a non-zero probability for all possible word strings, w_1^T. If there are *n* consecutive words extracted from a vocabulary of size V, there is the potential for $V^n - 1$ degrees of freedom [4]. When dealing with a natural language in a discrete space, the size of V can be rather large and in effect impractical, thereby requiring a reduced set of word sequences to be used for practical applications. Often, only those sequences of words that occur in the training data are considered rather than all possible combinations

of successive words [4]. Using a reduced set of word sequences in training the language model can result in a sparse data set. Therefore, word sequences in the application phase are likely to be different than those seen during training, resulting in incomplete representation of word sequences. This scenario in which the sparseness of the data results in incomplete representations of the word sequences is commonly referred to as the curse of dimensionality.

In an effort to combat the curse of dimensionality, Bengio et al [4] proposed modeling continuous variables in conjunction with of the use of *n*-grams. The *n*-gram models construct tables of conditional probabilities for the next word, for each one of a large number of combinations of the last *n*-1 words. The use of *n*-gram greatly reduces the complexity of the modeling problem by considering a sequence of *n* words at a time, using the *n*-1 words to predict the *n*th word instead of relying on the complete word history.

By inserting the conditional probability for the *n*-gram model shown in Equation 2.3 into Equation 2.2, the probability of the next word in the sequence based on the *n*-1 previous words can be determined [4].

$$\hat{P}\left(w_t|w_1^{t-1}\right) \approx \hat{P}\left(w_t|w_1^{t-n+1}\right) \tag{2.3}$$

In Equation 2.3, $\hat{P}\left(w_t|w_1^{t-n+1}\right)$ denotes the conditional probability based on the *n*-1 previous words instead of the complete word history $\hat{P}\left(w_t|w_1^{t-1}\right)$. By modeling with continuous variables, the function being learned (in this case through the use of multi-layer neural network) exhibits a smoothness property that cannot be observed with the discrete case [4]. By combining the use of the *n*-gram with the continuous variable approach, the effect of unseen *n*-grams in the word sequence is reduced and better generalizations are expected [2]. Schwenk's CSLM algorithm makes use of the *n*-gram simplification shown in Equation 2.3 in the application of a multi-layer neural network to perform the continuous space probability estimation for large vocabulary continuous speech recognition [3].

2.3 Neural Network Architecture

Artificial Neural Networks (ANNs) are mathematical models used to estimate functions that have a large number of inputs. The study of Artificial Neural Networks initially began as an attempt to understand the workings of the cognitive system in animals. ANNs differ from traditional computer solutions in that they are not programmed, but are trained to perform a function. The first attempts to model the neuron began in the 1940's, but the physical methods to do so at the time were purely hardware based [5]. The advancements in computing that followed the invention of the transistor allowed the transition of neuron modeling from hardware to software. The transition to modeling the networks on computers resulted in a more practical method to investigating the methods and applications of ANNs over the course of the following decades.

Presently, ANNs have found applications in function approximation, pattern recognition, classification, data processing, robotics, and control systems [5]. In the fields of pattern recognition and classification, ANNs have found practical applications in speech recognition, classification of handwritten characters, fault detection in machinery, and medical diagnosis [6]. A type of ANN, referred to as a perceptron, was coined by Rosenblatt [6] and was defined as a single layer network with threshold activation functions. Initially perceptrons were applied to classification problems where the inputs were binary images of simple shapes or objects. A single layer perceptron is very limited in its capabilities, failing to classify data sets that are not linearly separable. Further research into perceptrons showed that perceptron networks with just two layers were capable of approximating any continuous functional mapping. Multi-layered networks with either threshold or sigmoidal activation functions are called multi-layered perceptrons. The use of multi-layered perceptrons where the results at the output of the network can be interpreted as a probability density estimation is well documented [6].

The Continuous Space Language Model (CSLM) was introduced by Schwenk [3] along with an open source implementation of the algorithm [7]. The CSLM algorithm relies on a fully connected multi-layer perceptron neural network to perform the projection of the words into a continuous space and the probability estimation on this

space [7].The multi-layer perceptron neural network used in the algorithm consists of a projection layer, hidden layer, and output layer, as shown in Figure 2.1.

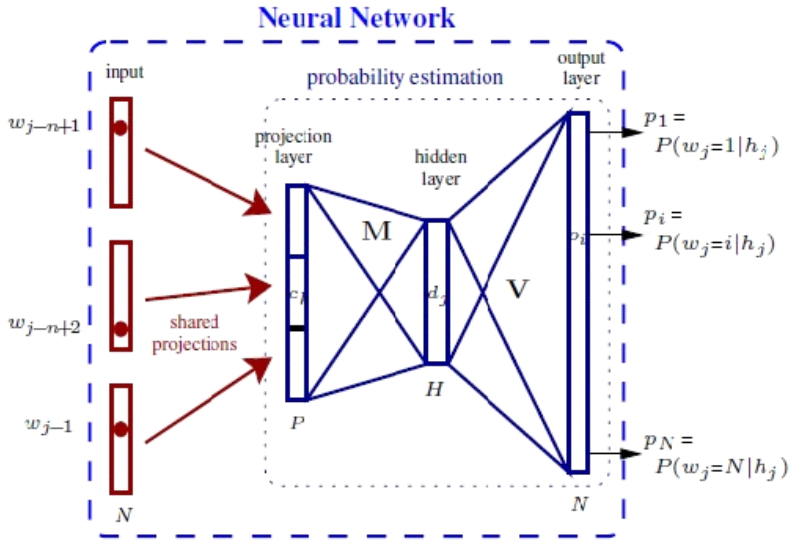

Fig. 2.1. Neural Network Structure [2]

The goal is to input an *n*-1 word sequence into the network, and the output of the network is the probability of all words in the vocabulary being the last, or *n*th, word in the sequence. These strings of *n* length sequences are referred to as *n*-grams. Schwenk's open source implementation of the CSLM algorithm provides everything required to train and evaluate the neural network. The neural network is trained through a process of adaptive learning using a large number of *n*-grams. This training phase of the CSLM algorithm was the focus of this research. The following paragraphs in this section provide a summary of the equations used in the ANN of the CSLM algorithm; the full explanation of these equations is well documented [1].

To start the training process, a list of vocabulary terms must be generated. This list is generally obtained using source text similar to that which will be encountered in the application phase. Schwenk's open source implementation includes two text files, news08.txt and news09.txt, which are combined into a text file vocab.txt, to form a list of vocabulary terms. Each of the resulting terms in vocab.txt is assigned a numerical index, which is used in computations for training the neural network [1].

The *n*-1 word sequence is input into the network at the projection layer. The numerical index previously assigned to each of the *N* words in the vocabulary corresponds to a row in the projection layer. Thus, the projection layer serves a look-up table in which each of the *N* rows contains a unique *P* length sequence for the corresponding word, where *P* is the user defined dimension of the continuous space. Given that Schwenk's open source implementation of the CSLM algorithm uses a 4-gram language model ($n = 4$), the indices of the first three words of the 4-gram are used to find, within the projection layer, the three unique *P* length sequences for that word. The concatenation of the *P* length sequences of the three word indices form the output of the projection layer. Thus, the output of the projection layer is a matrix, ***C***, formed by the row-wise concatentation of three matrices, each of size *P* x *X*, where *X* is the number of *n*-grams that are simultaneously input to the projection layer in batch mode. The resultant size of matrix ***C*** is 3*P* x *X*. This output of the projection layer is then used as the input to the hidden layer.

The hidden layer applies weights and biases to the output of the projection layer, followed by the hyperbolic tangent activation function to yield the hidden layer output. Equation 2.4, is the hidden layer output **D**, where **M** represents the hidden layer weight matrix of dimensions *H* x 3P and **B** is the bias matrix with dimensions *H* x *X*. *X* is the number of *n*-grams that are simultaneously input to the projection layer in batch mode, and *H* is the number of rows defined by the user in the hidden layer weight matrix. The biases are actually in a vector of size *H* x 1, which is copied *X* times to form the matrix **B** [1]. The output of the hidden layer, **D,** is a matrix of size *H* x *X*.

$$D = tanh(MC + B) \tag{2.4}$$

The hyperbolic tangent activation function is applied after the matrix multiplication and addition operations. It often results in faster convergence during training than logistic sigmoidal activation functions [6]. The use of the hyperbolic tangent as the sigmoidal activation function at the output of the hidden layer as part of the continuous conditional probability estimation in gradient based optimization algorithms is well documented [6]. The output of the hidden layer, **D**, is then passed on as the input to the output layer.

The output layer applies the associated weights and biases to the output of the hidden layer, **D**, to yield the result, **O**, of Equation 2.5, where **V** is the output layer weight matrix with dimensions *N* x *H* and **K** is the bias matrix with the dimensions *N* x *X*. Here, *N*, *H*, and *X* are the same as those defined above.

$$O = VD + K \tag{2.5}$$

The softmax normalization operation shown in Equation 2.6 is then applied to the output of Equation 2.5. The matrix **P** is the probability matrix for *X* number of *n*-grams and has the dimensions *N* x *X*

$$P_i = \frac{e^{O_i}}{\sum_{r=1}^{N} e^{O_r}} \tag{2.6}$$

In Equation 2.6, *i* is the column number for Matrix *O* of Equation 2.5, and *N* is the number of rows in matrix *O* (also the number of words in the vocabulary). The denominator of Equation 2.6 represents a summation over all rows in a given column of the matrix [1]. The use of the softmax activation function at the output of neural networks in conditional probability density estimation is well documented [6].

2.4 CSLM Training

In order to use the ANN, it must first be trained through a process of adaptive learning where the error between the output of the ANN and the desired response is minimized. Adaptive learning with ANNs consists of determining the values of the projection layer as well as the values of the weight and bias values for the ANN in order to minimize the error between the output of the ANN and the desired response. The training process consists of a forward pass of the input data through the layers of the ANN, followed by a backward pass in the reverse order.

In the forward pass, weights, biases and activation functions are applied to the input data through the use of Equations 2.4 - 2.6. In the backward pass, errors are propagated in the reverse direction to improve the weights and biases as well as update the projection layer. The equations used in the backward pass are not provided here but are described in detail by Thompson et al. [1].

At the completion of the forward pass, the backward pass starts at the output layer and progresses in the reverse direction through the layers of the ANN. The learning rate, λ, is updated for the present block of data being processed and is given by Equation 2.7, where λ_0 is the initial learning rate set by the user, *b* is the number of 4-grams processed thus far, and *r* is the learning rate multiplication factor set by the user. The value of *b* increments in multiples of the *X* number of 4-grams being processed in batch mode.

$$\lambda = \frac{\lambda_0}{(1+br)} \tag{2.7}$$

If the learning rate λ is too slow, the error reduction will be very slow, and if λ is too large, divergent oscillations may occur [5]. For a detailed explanation of the training process for the CSLM algorithm, the reader is referred to Thompson et al. [1].

The completion of a forward pass and backward pass over all *n*-grams constitutes what is called an epoch. At the completion of the epoch, a validation stage is implemented which consists of only the forward pass, without updates to weights, biases, or projection layer values. The purpose of the validation stage is to assess what the error would be if the training were stopped and the current weights, biases, and projection layer entries were used [1]. The process of training and validation continues for additional epochs until a desired perplexity level has been attained or the error has been reduced below a desired level. The training of the ANN is the most computationally intensive portion of Schwenk's CSLM algorithm, and its implementation on the Tegra K1 was the focus of this research.

3. GRAPHIC PROCESSING UNIT

3.1 Overview

The Graphics Processing Unit (GPU) is the core component of graphics cards found in laptops, PCs, and desktop computers. It is a specialized parallel device that was originally developed to meet the extreme computational demands of the real time video and computer gaming markets. Over time, developers realized that the GPU could also be used in non-graphic applications to improve the performance of computationally intensive algorithms through the use of the GPU's parallel processing architecture [1]. In an effort to further the use of GPUs in graphical and non-graphical applications, NVIDIA Corporation developed the Compute Unified Device Architecture (CUDA) for use with their GPU hardware. CUDA provides programmers with a method to make use of the parallel processing capabilities of the GPU by extending popular programming languages to run on the GPU hardware [1].

3.2 Architecture

The NVIDIA GPU architecture consists of a scalable array of multi-threaded Streaming Multiprocessors (SMs), each containing a number of processors. Figure 3.1 shows a block diagram of a NVIDIA GPU designed for a typical desktop application, such as the Quadro FX 5800 reported by Thompson et al. [1]. The Quadro FX 5800 contains 30 SMs, each containing 8 processors, for a total of 240 processors (a.k.a. cores). In contrast, the architecture of the NVIDIA Tegra K1 GPU designed for embedded applications is shown in Figure 3.2. The Kepler architecture features the advanced Streaming Multiprocessor (SMX) for higher performance and power efficiency

compared to that of the SM [8]. The GPU used in the Tegra K1 is a single Kepler SMX, containing 192 cores. The architecture of the Kepler GPU in Tegra K1 is virtually identical to the Kepler GPU used in high-end systems, but also includes optimizations for mobile system usage to conserve power and deliver mobile GPU performance [9].

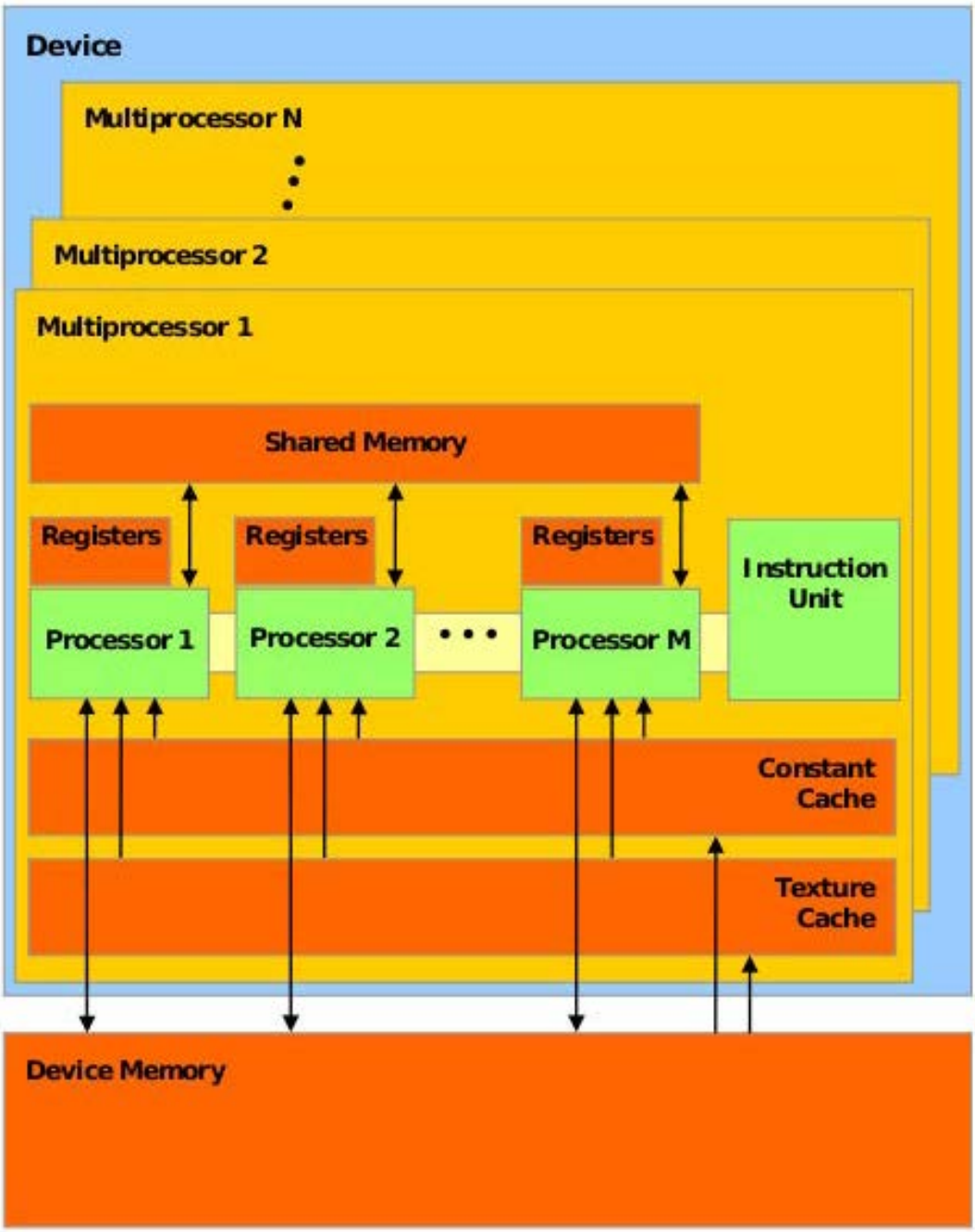

Fig. 3.1. General GPU Architecture [1]

The functional units within the SMX are a combination of single precision processing cores (total of 192), 32 Load/Store (LD/ST) units, 32 Special Function Units (SFUs), and 64 Double Precision Units (DP Units) [10].

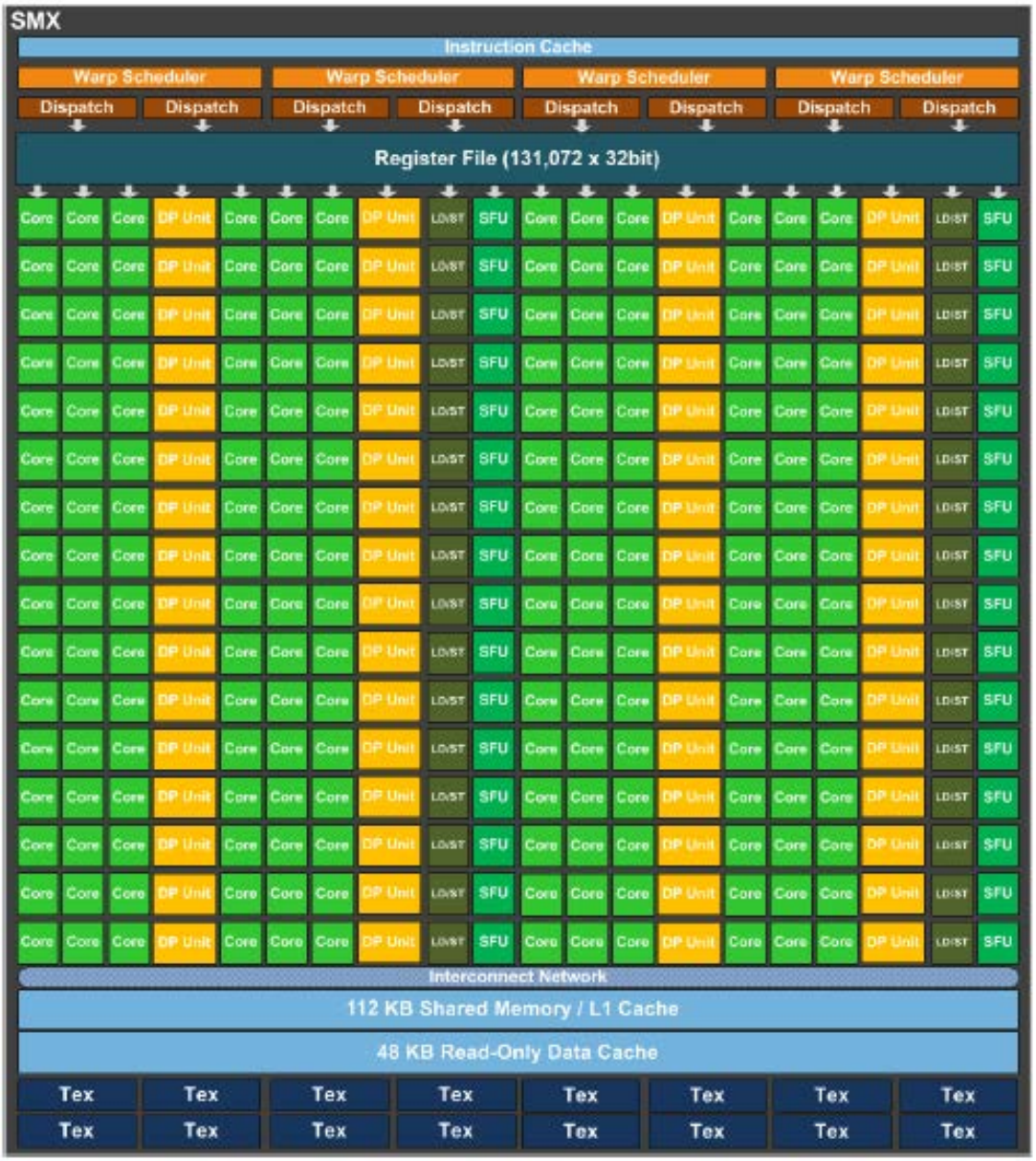

Fig. 3.2. Kepler SMX [10]

Along with the functional units within the SMX, each SMX contains an Instruction Cache, 4 Warp Schedulers, 8 Instruction Dispatch Units, a Register File, Shared and L1 Cache Memory, as well as some additional memory/resources. The GPU also contains L2

cache for system and GPU memory, memory controllers for the GPU memory, and a Host Interface for communication with the CPU [10].

Each of the Kepler SMX units features 192 single-precision CUDA cores, and each core consists of a pipelined Floating Point Unit (FP Unit) and an Integer Arithmetic Logic Unit (INT Unit) providing Fused Multiply Add (FMA) instructions for floating point operations [10].

In a typical desktop system, the GPU is a peripheral device that interfaces to the CPU(s) through a Peripheral Component Interconnect (PCI) bus or PCI Express (PCIe) bus. The GPU card within the desktop computer also provides the GPU with its own physical memory for loading and executing the GPU program and is accessible to both the CPU and the GPU. Data to be used by the GPU must be transferred from the CPU to the GPU. After performing computations on the GPU, the data must be transferred from the GPU back to the CPU. These data transfers are typically a bottleneck in the process. In contrast, the Tegra K1 integrates the CPU and the GPU on the same die, and provides both processors with access to shared memory. The shared memory allows the GPU within theTegra K1 to take advantage of the Unified Memory model. The Unified Memory model is a component of the CUDA programming model that defines a new managed memory space in which all CPU and GPU processors in the system see a single coherent memory image with a common address space [11]. This simplifies GPU programming and maximizes data access speed by transparently migrating data towards the processor using it and eliminates the need for explicit memory transfers between host and device [11]. The underlying system handles the memory copies in a transparent manner.

3.3 Implementation (CUDA)

Through the use of programming language extensions (CUDA in the context of this paper), programmers define the kernel functions that are to be called from the CPU (referred to as the host) program during execution on the GPU (referred to as the device). A kernel is a function that can be executed *N* times in parallel by *N* different threads. Each thread is a computation that is executed on the GPU by a processing core. Multiple

threads are grouped into blocks (called thread blocks) which, in turn, are grouped into grids. Within NVIDIA devices, grid dimensions are represented by a 3 dimensional coordinate system and block dimensions are represented through the use of a 3 dimensional coordinate system. Figure 3.3 shows the organization of threads, grids, and block in CUDA programming model.

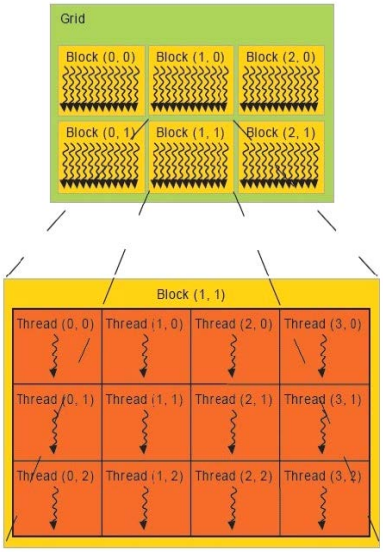

Fig. 3.3. NVIDIA Thread [11]

When writing the kernel function, the programmer defines the thread and block dimensions through the use of a three component vector. The three component vector allows the programmer the ability to define the block(s) within the grid(s) in one, two, or three dimensions so that there is a thread for each computation. The total number of threads for a kernel can be calculated based on the programmer defined grid and block dimensions. The number of threads used by a kernel is the product of the dimensions of the block times the number of blocks in the grid. There is a limit to the number of threads that can exist in a block, as well as the number of blocks in a grid. This is due to the fact that all of the threads in the block are required to reside on the same SM, and the number of SMs varies depending on the device.

NVIDIA has a number of different versions of GPU devices with varying performance capabilities, so a parameter called compute capability was created to define

the performance capabilities of the streaming multiprocessor in the different versions of devices. There are currently devices with compute capability versions ranging from 1.x to 5.x, with devices with higher compute capability exhibiting enhancements over the previous versions. The differences between all of the versions is outside the scope of this paper, so only the compute capabilities of the devices relative to the work discussed is this paper will be compared in the later sections.

In general, when a kernel is called from the host, the input data is copied over the interface bus from the CPU to the GPU memory. In the case of devices that support Unified Memory (devices that support CUDA 6.0 and higher), explicit memory copies are no longer required [11]. The underlying system manages memory copies in a transparent manner, hidden from the user. In architectures in which the CPU and the GPU are integrated on the same die and have access to shared memory, data transfer overhead could potentially be eliminated or reduced. After the GPU program is loaded over the interface bus, the GPU executes one or more grid blocks. One or more thread blocks are assigned to a SM; they are partitioned into groups of 32 threads called warps. If all 32 threads in a warp are not used, they are inactive, and the full capability of the GPU is not realized. To ensure that the maximum performance of the kernel can be achieved, the programmer needs to ensure that the number of threads is a multiple of 32 [11]. The warps are then scheduled for execution and passed to the processing elements through the warp dispatch units. The processing unit executes the threads and once the kernel execution has completed, the data is then copied back over the interface bus to the CPU memory for further use.

4. TEGRA K1 HETEROGENEOUS MOBILE PROCESSOR

NVIDIA introduced the Tegra K1 mobile processor in 2014 as the next generation in their Tegra family of mobile processors. The Tegra K1 is a heterogeneous mobile processor meaning that it consists of more than one kind of processor. The Tegra K1 contains both a GPU and a CPU. In a heterogeneous desktop environment involving a CPU and a GPU, the CPU resides on the motherboard, and the GPU is a card on a PCIe port on the motherboard, allowing the CPU to offload computations to the GPU. In this configuration, the CPU is required to transfer the data and the kernel to the GPU memory over the PCIe interface for execution and then transfer the data back from the GPU for use on the CPU once the kernel is complete. These data transfers are typically the bottleneck in the process. With multi-core CPU processors considered standard even in mobile processor devices, heterogeneous computing using GPUs have started to evolve onto mobile processors as well. In the Tegra K1 heterogeneous mobile processor, the GPU resides on the same chip as the CPU processor(s) with shared external memory, providing the mobile processor(s) the capability to offload computationally intensive functions to the GPU within the device, often avoiding costly overhead in data transfers between the CPU and GPU.

The 32-bit version of the Tegra K1 features a CPU Complex that consists of four 32-bit ARM Cortex-A15 “r3” processors running at 2.32 GHz, plus an additional fifth battery saver core for low power/light load operations [12]. In addition to the CPU Complex, the Tegra K1 features an NVIDIA Kepler GK20a Mobile GPU operating at 852 MHz (Appendix C). The Tegra K1 memory controller provides a 64-bit DRAM interface, capable of providing the Tegra K1 with up to 8 GB of DDR3L/LP.

The Kepler GK20a Mobile GPU consists of a single Kepler SMX with 192 CUDA cores, 64 Double Precision Units, 32 Special Function Units (SFUs), 32 load/store units,

and a compute capability of 3.2 [13]. Table 4.1 compares features of the Tegra K1 Kepler Mobile GPU to those of the Quadro FX GPUs used by Thompson et al. [1]. Although the Tegra K1 Mobile Kepler GPU has only a single advanced Streaming Multiprocessor, it features a higher compute capability than that of the Quadro FX GPU cards used by Thompson et al. [1]. The compute capability is a metric used by NVIDIA to classify the architecture of the device by the version number, and the version of the SM(X) by the revision [11]. The compute capability of a device specifies parameters such as maximum threads per block and maximum threads per SM(X) thereby corresponding to improvements in features between versions/revisions.

Table 4.1
Tegra K1 and Quadro FX Comparison

	Kepler Mobile (GK20A) Appendix C	Quadro FX 380 LP [1]	Quadro FX 2700 M [1]	Quadro FX 5800 [1]
Compute Capability	3.2	1.2	1.1	1.3
Streaming Multiprocessors (SM)*	1	2	6	30
CUDA Cores	192	16	48	240
Warp Size	32	32	32	32
Max Threads per SM	2048	1024	768	1024
Max Threads per Block	1024	512	512	512

*The Kepler GK20A contains the advanced SMX unit rather than the SM contained in the other GPU devices listed.

5. LAPTOP IMPLEMENTATION

5.1 Overview

For a basis of comparison of the performance of the CSLM algorithm on the Tegra K1 mobile processor, a CPU implementation of the CSLM algorithm was first compiled and executed on a laptop environment without a CUDA capable GPU. The CSLM algorithm as originally implemented by Schwenk is a CPU version that provides the option of incorporating high performance freely available BLAS or ATLAS libraries to accomplish fast matrix multiplications. Alternatively, it provides the option of utilizing the Intel MKL libraries, for a small fee. A more recent version of Schwenk's CSLM code incorporates support for various desktop GPUs, but as of the latest version (version 4 June 2015), the GPU on the Tegra K1 (mobile processor platform) is not supported. For the laptop CPU implementation reported here, BLAS libraries were the chosen option. The BLAS libblas libraries included in the Ubuntu 14.04 LTS [14] install were used in conjunction with version 1.0 of Schwenk's CSLM algorithm [15]. Only one of the two available CPUs of the laptop was used, with the CPU executing the algorithm using a single thread. Results of this implementation are reported in Section 5.8.

A second CPU implementation was accomplished on the ARM A15 CPU processors of the Tegra K1. Since the ARM A15 on the Tegra K1 consists of 4 CPUs, a multicore version of Schwenk's CSLM algorithm was executed on this platform. BLAS was again the chosen library option; however, the ARM A15 CPU processors present in the Tegra K1 are incompatible with the BLAS version used for the laptop CPU implementation [16]. Therefore, OpenBLAS version openblas-v0.2.8-armv7-rc2-.tar.gz was downloaded to the Tegra K1 ARM A15 processors from SourceForge.net [17] for the Tegra K1 multi-core CPU implementation of the CLSM algorithm and used in conjunction with version

1.0 of Schwenk's CSLM code. Two of the four ARM A15 processors were used in executing the code. Results of this implementation are reported in Section 6.8.

A third CPU implementation was subsequently executed on the laptop using the OpenBLAS version openblas-arm-devel-v0.2.8-rc2-src.tar.gz downloaded to the laptop from SourceForge.net [18]. The purpose was for direct performance comparison to that of the multicore CPU processor version on the Tegra K1. This implementation was also a multi-core CPU approach, and both of the two available CPU processors on the laptop were used. Version 1.0 of Schwenk's CSLM code was used, and results are reported in Section 5.8.

As mentioned previously, this effort focuses only on the training and validation portions of Schwenk's open source CSLM algorithm, consisting of computationally intensive operations performed for training and validation of the neural network. The training and validation portions of the algorithm making use of Equations 2.4 - 2.6 from Section 2.3 and described in Section 2.4 as well as the backward pass computations described in Thompson et al. [1] encompass the operations of the cslm_train executable from Schwenk's CSLM toolkit. The training and validation portion of Schwenk's CSLM toolkit is all that was run for the Tegra K1 implementation. For the OpenBLAS and GPU implementations on the Tegra K1, Schwenk's cslm_eval, net_info, and text2bin executables were not run. All reference to execution times in this paper are with respect to the epoch execution times displayed during the execution of the cslm_train executable.

The following sections describe the laptop test platform as well as details of the BLAS, MKL, and OpenBLAS libraries. A description of the installation of the OpenBLAS libraries for the laptop implementation is also provided. The description of the implementation of OpenBLAS for the Tegra K1 is discussed in Section 6.3.

5.2 Test Platform

The laptop used in this study is a Sony VPCCA290X featuring two Intel i5-2410M Sandy Bridge CPUs and 16 GB of DDR3 SDRAM running 64-bit Linux Ubuntu LTS 14.04 operating system [19] [14]. Because this laptop does not feature a NVIDIA CUDA capable graphics processor, the version the CSLM algorithm used on this platform will

use only the CPU processors. Table 5.1 lists the specifications of the Intel i5-2410M processors of the Sony VPCCA290X laptop used to run the CSLM algorithm.

Table 5.1
VPCCA290X Laptop Processor Specifications [19]

Processor	Intel i5-2410M
Processing cores	2
Threads	4
Base frequency	2.3 GHz
L1 Cache	64 KB
L2 Cache	256 KB
Memory	DDR3 SDRAM
Size	16 GB
Maximum Memory Transfer Rate	21.3 GB/s

5.3 BLAS

Basic Linear Algebra Subprograms (BLAS) package is a freely available software package first developed in FORTRAN to provide a set of low level routines for linear algebra operations [20]. Since its initial release, BLAS has been updated to support C and FORTRAN languages. In addition, it has been optimized for particular CPU/GPU architectures by manufacturers such as Intel, AMD, HP, and NVIDIA. The Ubuntu Linux distribution includes the libblas BLAS libraries in its library as a standard package for performing basic vector and matrix operations. BLAS functions are divided into three levels, where each level performs certain operations: Level 1 BLAS performs scalar, vector, and vector-vector operations, Level 2 BLAS performs vector-matrix operation, and Level 3 BLAS performs matrix-matrix operations. Each level of BLAS supports single, double, complex, and double complex numerical precision. In Schwenk's CSLM code, if the BLAS option is selected, BLAS is used to perform the matrix multiplication operations of Equations 2.4 and 2.5 in Section 2.3 as well as the backward pass matrix multiplications described by Thompson et al. [1]. It should be noted here that the ARM A15 processor present in the Tegra K1 does not currently support BLAS [16].

5.4 MKL

Intel's Math Kernel Library (MKL) is a library developed by Intel and optimized to run on Intel multiple CPU architectures to increase performance of matrix, vector, linear algebra, and statistical functions. Schwenk's CSLM algorithm supports the use of MKL through the use of pre-compiler directives to perform the Level 3 BLAS General Matrix Multiplication (GEMM) functions performed in forward and backward pass.

$$C = \alpha AB + \beta C \tag{5.1}$$

The GEMM functions calls perform the operation shown in Equation 5.1 where *A*, *B*, and *C* are matrices and α and β are scalars. Although Thompson et. al. [1] compared the performance of the CLSM algorithm using the MKL libraries to a GPU based version, the Intel MKL libraries are incompatible with the ARM A15 processors residing on the Tegra K1; therefore, MKL was not investigated in this work.

5.5 OpenBLAS

OpenBLAS is an open source optimized BLAS library forked from GotoBLAS2-1.13 which is another open source software library that was released by Texas Advanced Computing Center. When the main developer, Mr. Kazushige Goto, left Texas Advanced Computing, OpenBLAS was created to continue developing OpenBLAS/GotoBLAS [21]. OpenBLAS provides optimized implementations of linear algebra kernels for several processor architectures (including ARM A15) and achieves performance similar to those of MKL [22]. OpenBLAS was the only available version of BLAS compatible with the ARM A15 multicore CPU processors of the Tegra K1 and was therefore used on this platform. In addition, a multicore version of OpenBLAS was also implemented on the laptop version of the CSLM algorithm for direct performance comparison to that of the multicore CPU processor version on the Tegra K1. When implementing the CSLM algorithm using OpenBLAS on the Tegra K1, two of the ARM A15 CPU processors were used; on the comparable laptop implementation, both of the two Intel i5-2410M Sandy Bridge CPUs were used.

To install OpenBLAS on the Sony VPCCA290X laptop, the source version openblas-arm-devel-v0.2.8-rc2-src.tar.gz of OpenBLAS for the x86 architecture of the laptop was downloaded from SourceForge.net [18]. Since OpenBLAS also requires the installation of the libgfortran library, Synaptic Package Manager was first installed from the Ubuntu Software Center from within Ubuntu 14.04 LTS [23]. From Synaptic Package Manager, versions libgfortran3 and libgfortran-4.8-dev were downloaded and installed on the laptop. This placed the libgfortran archive file, libgfortran.a, and the shared link file, libgfortran.so, at the path location shown in Figure 5.1.

```
/usr/lib/gcc/x86_64-linux-gnu/4.8
```

Fig. 5.1. Default path of the libgfortran.a and libgfortran.so files after installation of the libgfortran library

The shared link file libgfortran.so provides the reference to the shared link file libgfortran.so.3 which is located at the directory shown in Figure 5.2.

```
/usr/lib/x86_64-linux-gnu
```

Fig. 5.2. Default location of the shared link file libgfortran.so.3 and shared library file libgfortran.so.3.0.0

The shared link libgfortran.so.3 file provides the reference to the shared library file libgfortran.so.3.0.0. OpenBLAS was built for the laptop by navigating from the terminal to the location of the source directory, OpenBLAS-devel-v0.2.8-rc2, shown in Figure 5.3 and then running the command shown in Figure 5.4 at the terminal prompt. This command creates libopenblas.a archive and the libopenblasp-r0.2.8.so shared object for the laptop. The command TARGET=SANDYBRIDGE BINARY=64 DYNAMIC_ARCH=1 NUM_THREAD=4 shown in Figure 5.4 provides the makefile with directives to build OpenBLAS for the Sandy Bridge 64-bit architecture of the laptop using all 4 threads available on the laptop through the use of Intel's Hyper Threading Technology (HTT) [19]. The HTT feature of the Intel i5-2410M processor delivers two processing threads per physical core [19].

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/OpenBLAS-devel-v0.2.8-rc2$
```

Fig. 5.3. Source directory location for OpenBLAS build makefile

Upon the successful completion of the build as shown in Figure 5.5, the newly created files libopenblas.a and libopenblasp-r0.2.8.so by default were located in the directory in which the build occurred, and were copied to the /usr/lib location where the OpenBLAS version of the algorithm is set to reference.

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop$ make TARGET=SANDYBRIDGE BINARY=64
DYNAMIC_ARCH=1 NUM_THREAD=4
```

Fig. 5.4. OpenBLAS laptop build command

```
OpenBLAS build complete. (BLAS CBLAS LAPACK LAPACKE)

 OS             ... Linux
 Architecture    ... x86_64
 BINARY          ... 64bit
 C compiler      ... GCC  (command line : gcc)
 Fortran compiler ... GFORTRAN  (command line : gfortran)
 Library Name    ... libopenblasp-r0.2.8.a (Multi threaded; Max num-threads is 4)

To install the library, you can run "make PREFIX=/path/to/your/installation install".

kurt@kurt-VPCCA290X: /home/kurt/cslm_laptop /cslm_laptop/OpenBLAS-devel-v0.2.8-rc2$
```

Fig. 5.5. Laptop OpenBLAS Succesful Build

Subsequently the symbolic linkslibopenblas.so and libopenblas.so.0 were created as shown in Figure 5.6 to link to the libopenblasp-r0.2.8.so shared library object to provide the link paths to the libopenblasp-r.0.2.8.so file.

```
kurt@kurt-VPCCA290X:/usr/lib$sudo ln -s libopenblasp-r0.2.8.so libopenblas.so
kurt@kurt-VPCCA290X:/usr/lib$sudo ln -s libopenblasp-r0.2.8.so libopenblas.so.0
```

Fig. 5.6. OpenBLAS Symbolic Link Commands

5.6 BLAS Implementation on the SONY VPCCA290X Laptop

As discussed in Section 5.1, the BLAS libraries packaged with the Ubuntu Linux operating system [14] were automatically available on the Sony VPCCA290X laptop upon installation of the operating system. Schwenk's CSLM open source toolkit provides all of the source files to compile all the libraries and generate the executable required to run the algorithm. The makefile provided by Schwenk's CSLM open source toolkit version 1.0 [15] provides the options for use of either BLAS or MKL libraries, so this makefile required some modification before the algorithm could be run on the Sony VPCCA290X laptop using BLAS. In Schwenk's code, the makefile is located in the directory /src. Lines 28 and 33 of this makefile are shown in Figure 5.7. By default, this makefile specifies use of the MKL libraries. To specify instead the BLAS library, line 28 of the makefile was uncommented, and line 33 was commented out (as denoted by the leading # sign) as shown in Figure 5.8 to select libblas libraries in lieu of the MKL libraries.

```
28#BLAS=-DBLAS_STD
33BLAS=-DBLAS_INTEL_MKL -I/opt/intel/Compiler/11.1/046/mkl/include.
```

Fig. 5.7. Original CSLM Makefile BLAS Library Setting

```
28BLAS=-DBLAS_STD
33#BLAS=-DBLAS_INTEL_MKL -I/opt/intel/Compiler/11.1/046/mkl/include
```

Fig. 5.8. Updated CSLM Makefile BLAS Library Setting to select BLAS libraries rather than MKL

The library path locations specified in the makefile were then modified from the default locations shown in Figure 5.9 to point to the location of the BLAS library. This was accomplished by commenting out lines 34 and 35 of the makefile, uncommenting line 29 and changing the file and path on line 29 to the location of the BLAS libraries, as shown in Figure 5.10.

```
34LIBS_INTEL=/opt/intel/Compiler/11.1/046
35LIBS_MKL=-L$(LIBS_INTEL)/mkl/lib/em64t -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -
L$(LIBS_INTEL)/lib/intel64 -liomp5 -lpthread -lm
29#LIBS_MKL=/usr/lib64/libblas.so.3
```

Fig. 5.9. Original CSLM Makefile BLAS library location

```
34#LIBS_INTEL=/opt/intel/Compiler/11.1/046
35#LIBS_MKL=-L$(LIBS_INTEL)/mkl/lib/em64t -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -
L$(LIBS_INTEL)/lib/intel64 -liomp5 -lpthread -lm
29LIBS_MKL=/usr/lib/libblas.so
```

Fig. 5.10. Updated CSLM Makefile BLAS library location

Note that the complete original CSLM makefile is shown in Appendix A.1, and the complete modified makefile, incorporating all the modifications discussed above is provided in Appendix A.2.

In addition to using the BLAS libraries, Schwenk's CSLM open source toolkit also utilizes the SRI Language Modeling (SRILM) toolkit from SRI International [24]. The SRILM toolkit is used by Schwenk's open source CSLM toolkit to generate the data files used by the cslm_train executable. The cslm_train executable is generated upon completion of the build of the CSLM toolkit and is located in the /src/ directory. The cslm_train executable performs the training and validation portions of the algorithm described in Section 2.4. SRILM is available only in source form. The source code must be downloaded, and then the software must be built and installed. SRILM version 1.6 was downloaded to the Sony VPCCA290X laptop from the SRI International website [24]. It should be noted that initially, SRILM version 1.7.1 was attempted; however, there were compatibility issues with CSLM verion 1.0, so SRILM version 1.6 was used instead. Before the SRILM could be built by executing the makefile, a modification was made to the makefile to specify the correct location of the SRILM directory. Figure 5.11 shows line 7 of the original SRILM makefile pointing to the default location of the SRILM source directory, and Figure 5.12 points to the location on the laptop after the modification. Appendix A.4 lists the full original SRILM makefile; Appendix A.5 lists the full SRILM makefile with the modifications.

```
7# SRILM = /home/speech/stolcke/project/srilm/devel
```

Fig. 5.11. Original Line 7 of SRILM makefile

```
7SRILM = /home/kurt/Desktop/cslm_laptop/srilm-1.6.0
```

Fig. 5.12. Modification of Line 7 of SRILM makefile

The SRILM toolkit was then compiled on the Sony VPCCA290X laptop by navigating at the terminal to the directory containing the makefile, as shown in Figure 5.13, and executing the command shown in Figure 5.14.

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/srilm-1.6.0$
```

Fig. 5.13. Directory location of the SRILM makefile

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/srilm-1.6.0$make MACHINE_TYPE=i686-m64 World
```

Fig. 5.14. SRILM makefile command

Figure 5.15 shows the results of a successful build of the SRILM libraries after executing the terminal command shown in Figure 5.14. The completion of the build creates the oolm.a, dstruct.a, misc.a, z.a, and m.a libraries at /home/kurt/cslm_laptop/srilm-1.6.0/lib/i686-m64.

```
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 make-batch-counts
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 merge-batch-counts
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 make-big-lm
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 make-multiword-pfsg
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 pfsg-from-ngram
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 nbest-error
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 nbest-rover
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 align-with-tags
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 compute-sclite
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/sbin/decipher-install 0555 compare-sclite
/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/srilm-1.6.0$
```

Fig 5.15. SRILM Build Results

The build process resulted in the creation of three directories: bin/, lib/, and include/. Having compiled the SRILM toolkit, the CSLM open source toolkit makefile was updated to point to the the location of the SRILM directory on the laptop. Line 17 was added as shown in Figure 5.16 to point to the required SRILM header files used in the CSLM build.

```
17SRILM = /home/kurt/Desktop/cslm_laptop/srilm-1.6.0
```

Fig. 5.16. Path setting in CSLM makefile to point to SRILM Header Files

Having compiled the SRILM and updated the CSLM makefile to point to the libblas BLAS libraries, the CSLM makefile was ready to be run on the laptop from the terminal by typing the “make” command in the terminal at the cslm source directory location as shown in Figure 5.17.

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/cslm_v1.0/src$ make
```

Fig. 5.17. CSLM BLAS makefile execution command

It should be noted here that DataNgramBin.cpp had to be modified to resolve a compile error relating to not recognizing the read, close, and lseek functions used in DataNgramBin.cpp. Figure 5.18 shows the addition of the #include <unistd.h> at line 25 of DataNgramBin.cpp, which was added to enable DataNgramBin.cpp to compile on the laptop.

```
23using namespace std;
24#include <iostream>
25#include <unistd.h>
```

Fig. 5.18. Addition of #include<unistd.h> to DataNgramBin.cpp

When the makefile completes, executables for cslm_eval, cslm_train, net_info, text2bin, and the archive libcslm.a will have been generated. The cslm_eval executable is used to evaluate the perplexity at the completion of the training and validation accomplished by the cslm_train executable. The text2bin executable is used to binarize the list of vocabulary terms for use in training and validation. The net_info executable is used to inspect the saved network upon completion of the training and validation accomplished by the cslm_train executable. The cslm_train executable performs the training and validation as described in Section 2.4 using Equations 2.4 - 2.6 described in Section 2.3 for the forward pass as well as the operations described by Thompson et al. [1] for the backward pass.

Before the CSLM algorithm could be executed, the original run.csh file located in the “example/ngram/” subdirectory of the cslm_v1.0 directory and shown in Figure 5.19 had to be modified at lines 9-12 to point to the location of the executables from the CSLM build, as shown in Figure 5.20**,** in which the revisions are shown in red.

```
#!/bin/csh
#
# This is a short example how to use the CSLM toolkit
#
# Data (tokenized with default scripts):
#  - news08.txt, news09.txt       data from WMT'10
#

# where to find the CSLM binaries
set text2bin=../../text2bin
set cslm_train=../../cslm_train
set cslm_eval=../../cslm_eval
set net_info=../../net_info
# specify where to find the BLAS libraries
setenv LD_LIBRARY_PATH
/opt/intel/Compiler/11.1/046/mkl/lib/em64t:/opt/intel/Compiler/11.1/046/lib/intel64
setenv OMP_NUM_THREADS 8  # set to the number of CPU cores you have

# get the word list
cat news0[89].txt | ngram-count -unk -order 1 -text - -write /tmp/counts.gz
zcat /tmp/counts.gz | sort -nr -k2 | awk '{print $1}' | sort > vocab.txt
/bin/rm /tmp/counts.gz

echo "Build simple back-off LM for comparision"
 ngram-count -unk -order 4 -text news08.txt -vocab vocab.txt \
        -interpolate -kndiscount -gt1min 1 -gt1min 1 -gt1min 2 -gt1min 3 \
        -lm lm.arpa.gz
ngram -order 4 -unk -lm lm.arpa.gz -ppl news09.txt

echo "Binarizing data"
foreach f (news08 news09)
  cat $f.txt | $text2bin vocab.txt $f.btxt $f.wlist $f.oov
end

#
# Start training
#   feel free to adapt the network architecture, learning rate, etc
#
# You should get a result similar to:
# Starting epoch 10 at Mon Jan 25 11:39:36 2010
#  - intial lrate=3.4897e-03, wdecay=3.0000e-05
#  - shuffling data 10 times ... done
#  - epoch finished, 60107 examples seen, average error: 105.625
#  - starting validation ... avrg error: 85.7966
# Starting epoch 10 at Thu Jan 28 14:47:44 2010
#  - intial lrate=3.4385e-03, wdecay=3.0000e-05
#  - shuffling data 10 times ... done
#  - epoch finished, 63070 examples seen, average error: 134.671
#  - starting validation ... avrg error: 109.356
# Training stopped
#
```

Fig. 5.19. Original run.csh

```
# Training takes about 7 min on a state-of-the-art server using Intel's MKL
# and multi-threading with 8 cores

$cslm_train train.df dev.df example.mach  256 192 14024 4 128   5e-3 4e-7 3e-5 10

#
# There is a tool to inspect saved networks
#
echo ""
$net_info example.mach

#
# You can also calculate the perplexity
# The result may be not directly comparable with the one of the SRILM toolkit
#  in function of the mode used (how to deal with UNK)
echo ""
$cslm_eval example.mach news09.btxt 4 3
```

Fig. 5.19. Continued

```
#!/bin/csh
#
# This is a short example how to use the CSLM toolkit
#
# Data (tokenized with default scripts):
#  - news08.txt, news09.txt       data from WMT'10
#

# where to find the CSLM binaries
set text2bin=../../src/text2bin
set cslm_train=../../src/cslm_train
set cslm_eval=../../src/cslm_eval
set net_info=../../src/net_info
# specify where to find the BLAS libraries
setenv LD_LIBRARY_PATH
/opt/intel/Compiler/11.1/046/mkl/lib/em64t:/opt/intel/Compiler/11.1/046/lib/intel64
setenv OMP_NUM_THREADS 8  # set to the number of CPU cores you have

# get the word list
cat news0[89].txt | ngram-count -unk -order 1 -text - -write /tmp/counts.gz
zcat /tmp/counts.gz | sort -nr -k2 | awk '{print $1}' | sort > vocab.txt
/bin/rm /tmp/counts.gz

echo "Build simple back-off LM for comparision"
 ngram-count -unk -order 4 -text news08.txt -vocab vocab.txt \
        -interpolate -kndiscount -gt1min 1 -gt1min 1 -gt1min 2 -gt1min 3 \
        -lm lm.arpa.gz
ngram -order 4 -unk -lm lm.arpa.gz -ppl news09.txt

echo "Binarizing data"
foreach f (news08 news09)
  cat $f.txt | $text2bin vocab.txt $f.btxt $f.wlist $f.oov
end

#
# Start training
#   feel free to adapt the network architecture, learning rate, etc
#
# You should get a result similar to:
# Starting epoch 10 at Mon Jan 25 11:39:36 2010
#  - intial lrate=3.4897e-03, wdecay=3.0000e-05
#  - shuffling data 10 times ... done
#  - epoch finished, 60107 examples seen, average error: 105.625
#  - starting validation ... avrg error: 85.7966
# Starting epoch 10 at Thu Jan 28 14:47:44 2010
#  - intial lrate=3.4385e-03, wdecay=3.0000e-05
#  - shuffling data 10 times ... done
#  - epoch finished, 63070 examples seen, average error: 134.671
#  - starting validation ... avrg error: 109.356
# Training stopped
#
# Training takes about 7 min on a state-of-the-art server using Intel's MKL
```

Fig. 5.20. Modified run.csh

```
# and multi-threading with 8 cores

$cslm_train train.df dev.df example.mach  256 192 14024 4 128   5e-3 4e-7 3e-5 10

#
# There is a tool to inspect saved networks
#
echo ""
$net_info example.mach

#
# You can also calculate the perplexity
# The result may be not directly comparable with the one of the SRILM toolkit
#  in function of the mode used (how to deal with UNK)
echo ""
$cslm_eval example.mach news09.btxt 4 3
```

Fig. 5.20. Continued

The path variable to the SRILM executable also had to be updated by typing at the terminal the command shown in Figure 5.21 so that calls to executables within the SRILM toolkit location could be made from the CSLM run.csh file.

```
PATH=$PATH:/home/kurt/Desktop/cslm_laptop/srilm-1.6.0/bin/i686-m64
```

Fig. 5.21. Update to path variables to include SRILM toolkit location

With the SRILM toolkit compiled, the CSLM toolkit compiled, and the run.csh file updated, the CSLM algorithm could be executed. Before running run.csh, it is recommended that clean_up.csh is executed first as shown in Figure 5.22 to remove the files from any previous runs.

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/cslm_v1.0/example/ngram$ ./clean_up.csh
```

Fig. 5.22. CSLM File Cleanup execution command

Table 5.2 shows the files deleted by the execution of the clean_up.csh. The CSLM algorithm is executed from the terminal by navigating to the directory in which it is located (/example/ngram) and typing “./run.csh” as shown in Figure 5.23.

Table 5.2
Files deleted by clean_up.csh

File
example.mach
lm.arpa.gz
vocab.txt
news09.oov
news08.oov
news08.btxt
news09.btxt
news08. Wlist
news09. Wlist

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/cslm_v1.0/example/ngram$ ./run.csh
```

Fig. 5.23. CSLM execution command

Upon execution of the run.csh script file in Schwenk's open source implementation, a list of vocabulary terms is generated. The vocabulary list used in training the algorithm is generally obtained using source text similar to that which will be encountered in the application phase. Schwenk's Ver. 1.0 open source implementation includes two text files, news08.txt and news09.txt, which are combined into a text file vocab.txt, to form a list of vocabulary terms. This operation occurs on lines 19 – 21 of the run.csh file of Figure 5.20. On lines 24 – 27 of Figure 5.20, a simple 4-gram backoff language model (LM) is then generated using the SRILM ngram-count and ngram executables. On lines 30 – 32 of Figure 5.20, the run.csh file invokes commands to binarize the news08.txt and news09.txt files using the text2bin executable from Schwenk's CSLM toolkit, resulting in the binary files news08.btxt and news09.btxt. Each of the corresponding terms in vocab.txt is assigned a numerical index, which is used to map to a row in the projection layer. On line 54 of Figure 5.20, run.csh then executes the cslm_train executable with the parameters shown in Figure 5.24. These command line parameters specify the use of file train.df for training and the file dev.df for validation. Upon completion, the trained neural network will be stored in example.mach. The remaining command line parameters specify 256 for the dimension of the projection layer, 192 for the dimension of the hidden

layer, 14024 for the dimension of the output layer, an order 4-gram, a block size of 128, a learning rate of 5e-3, a multiplication factor of 4e7 for the learning rate, a Wdecay value of 3e-5 (used to prevent the neural network from overfitting the training data), and the use of 10 epochs in training [3]. The cslm_train executable on line 54 of Figure 5.20 then proceeds to create the ANN as a unit consisting of a projection layer, followed by a hidden layer and an output layer, as depicted in Figure 2.1, and subsequently initializes this ANN with uniformly distributed random variables in the projection layer as well as in the weights (matrices **M** of Equation 2.4 and **V** of Equation 2.5) and biases (matrices **B** of Equation 2.4 and **K** of Equation 2.5) of the hidden and output layers.

```
$cslm_train train.df dev.df example.mach  256 192 14024 4 128   5e-3 4e-7 3e-5 10
```

Fig. 5.24. run.csh cslm_train execution

Once the initial values for the projection, hidden, and output layers have been generated, the training of the ANN can begin as described in Sections 2.3 and 2.4 above. Upon the completion of the cslm_train executable, run.csh then launches the net_info executable, line 60 of Figure 5.20, to inspect the saved networks after which the cslm_eval executable is run, line 67 of Figure 5.20, to calculate the perplexity. At this point the run.csh file has finished, and the BLAS version of the CSLM algorithm is complete. The text output from the execution of the run.csh is copied from the terminal and saved in a text file for a comparison later. An example of such output is shown in Figure 5.25. The output provides a timestamp for the start of each of the ten epochs used in training, enabling determination of execution time of the algorithm. It also provides a value for the average error at the completion of each epoch. Note that in the output of Figure 5.25, the training process completed in about 120 minutes, with an average error of 109.359 at the completion of the 10^{th} epoch.

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/cslm_v1.0/example/ngram$ ./run.csh
Build simple back-off LM for comparision
file news09.txt: 2525 sentences, 65595 words, 0 OOVs
0 zeroprobs, logprob= -186780 ppl= 551.986 ppl1= 703.843
Binarizing data
Text to binary converter V1.0, H. Schwenk, LIUM, University of Le Mans, France
 - using word list vocab.txt (14023 words, unk=0, bos=1, eos=2)
 - writing binary representation to file news08.btxt
 - dumping word frequencies to file news08.wlist
 - dumping list of OOV to file news08.oov
 - 2051 lines with 49765 words processed, 8427 uniq words (60.09% of the vocabulary)
 - 0 words were unknown ( 0.00% of the text), -4 new words
Text to binary converter V1.0, H. Schwenk, LIUM, University of Le Mans, France
 - using word list vocab.txt (14023 words, unk=0, bos=1, eos=2)
 - writing binary representation to file news09.btxt
 - dumping word frequencies to file news09.wlist
 - dumping list of OOV to file news09.oov
 - 2525 lines with 65595 words processed, 9726 uniq words (69.36% of the vocabulary)
 - 0 words were unknown ( 0.00% of the text), -4 new words
Sequential machine [3] 3- .. -14024, bs=128, passes=0/0
  Parallel machine 3- .. 768, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
  MachTanh 768-192, bs=128, passes=0/0
  MachSoftmax 192-14024, bs=128, passes=0/0
Opening data description 'train.df'
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Summary of used data:
 - news09.btxt  1.0000 *     63070 =     63070
 - total number of examples: 63070
 - resampling with seed 12345678
 - all resampling coefficients are set to one, loading data once
 - loading all data into memory
 - shuffling data 10 times ... done
Opening data description 'dev.df'
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Summary of used data:
 - news09.btxt  1.0000 *     63070 =     63070
 - total number of examples: 63070
 - resampling with seed 12345678
 - all resampling coefficients are set to one, loading data once
 - loading all data into memory
Starting training on host kurt-VPCCA290X pid 728
 - training on train.df
 - validation on dev.df
 - stopping training at 10 epochs
 - Sequential machine [3] 3- .. -14024, bs=128, passes=0/0
```

Fig. 5.25. Results of executing the CSLM algorithm on the Sony VPCCA290X laptop using the BLAS libraries

```
  Parallel machine 3- .. 768, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
  MachTanh 768-192, bs=128, passes=0/0
  MachSoftmax 192-14024, bs=128, passes=0/0
Starting epoch 1 at Sun Nov 15 19:08:30 2015
 - intial lrate=5.0000e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 950.157
 - starting validation ... avrg error: 566.79
Starting epoch 2 at Sun Nov 15 19:20:54 2015
 - intial lrate=4.7598e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 522.231
 - starting validation ... avrg error: 420.935
Starting epoch 3 at Sun Nov 15 19:33:17 2015
 - intial lrate=4.5417e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 407.593
 - starting validation ... avrg error: 332.252
Starting epoch 4 at Sun Nov 15 19:45:40 2015
- intial lrate=4.3427e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 335.863
 - starting validation ... avrg error: 285
Starting epoch 5 at Sun Nov 15 19:58:05 2015
 - intial lrate=4.1603e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 283.562
 - starting validation ... avrg error: 234.437
Starting epoch 6 at Sun Nov 15 20:10:27 2015
 - intial lrate=3.9927e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 242.75
 - starting validation ... avrg error: 199.291
Starting epoch 7 at Sun Nov 15 20:22:49 2015
- intial lrate=3.8381e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 208.62
 - starting validation ... avrg error: 170.99
Starting epoch 8 at Sun Nov 15 20:35:10 2015
- intial lrate=3.6950e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 179.751
 - starting validation ... avrg error: 144.582
Starting epoch 9 at Sun Nov 15 20:47:32 2015
 - intial lrate=3.5621e-03, wdecay=3.0000e-05
- shuffling data 10 times ... done
- epoch finished, 63070 examples seen, average error: 155.228
- starting validation ... avrg error: 125.407
```

Fig. 5.25. Continued

```
Starting epoch 10 at Sun Nov 15 20:59:54 2015
 - intial lrate=3.4385e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 134.674
 - starting validation ... avrg error: 109.359
Training stopped
 - Sequential machine [3] 3- .. -14024, bs=128, passes=1261400/630700
 -   Parallel machine 3- .. 768, bs=128, passes=1261400/630700
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -   MachTanh 768-192, bs=128, passes=1261400/630700
 -   MachSoftmax 192-14024, bs=128, passes=1261400/630700

Information on machine: example.mach
 - Sequential machine [3] 3- .. -14024, bs=128, passes=1261400/630700
 -   Parallel machine 3- .. 768, bs=128, passes=1261400/630700
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -   MachTanh 768-192, bs=128, passes=1261400/630700
 -   MachSoftmax 192-14024, bs=128, passes=1261400/630700

Evaluating CSLM: example.mach
 - Sequential machine [3] 3- .. -14024, bs=128, passes=1261400/630700
 -   Parallel machine 3- .. 768, bs=128, passes=1261400/630700
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -   MachTanh 768-192, bs=128, passes=1261400/630700
 -   MachSoftmax 192-14024, bs=128, passes=1261400/630700
Using data:
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
   counting ... 63070 4-grams (0 unk, 5050 ignored)
Perplexity: 109.356

kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/cslm_v1.0/example/ngram$
```

Fig. 5.25. Continued

5.7 OpenBLAS Implementation on the SONY VPCCA290X Laptop

Having compiled the OpenBLAS libraries as described in Section 5.5 above, the makefile provided by Schwenk's CSLM open source toolkit version 1.0 was modified to specify the use of OpenBLAS libraries rather than the BLAS version. To use the OpenBLAS libraries instead of the BLAS libraries, the library path location in the CSLM makefile was changed to point to the location on the laptop at which the OpenBLAS

library was located. This was accomplished by changing line 29 of the CSLM makefile from that shown in Figure 5.26 to that shown in Figure 5.27.

```
29LIBS_MKL=/usr/lib/libblas.so
```

Fig. 5.26. Original LIBS_MKL library location in the makefile of Schwenk's CSLM open source toolkit version 1.0

```
29LIBS_MKL=/usr/lib/libopenblas.so
```

Fig. 5.27. Modified CSLM Makefile LIBS_MKL library location to specify OpenBLAS libraries

The makefile also required an addition to line 41 from the default as shown in Figure 5.28 to the modified version of Figure 5.29 to include the referenced header files for OpenBLAS.

```
CFLAGS=-Wall -I$(SRILM)/include -g ${DB} ${BLAS}
```

Fig. 5.28. Default line 41 of CSLM Makefile

```
CFLAGS=-Wall -I$(SRILM)/include -I/home/kurt/Desktop/Thesis_2015/software/OpenBLAS-devel-v0.2.8-rc2 -g ${DB} ${BLAS}
```

Fig. 5.29. Modified Line 41 of CSLM Makefile to include OpenBLAS header

The modifications to the CSLM makefile in Figures 5.7-5.10, and the change to DataNgramBin.cpp in Figure 5.18 for the BLAS version are still valid, and no other changes are required to the makefile. Before executing the CSLM makefile, the previous version of the cslm_train executable should be deleted if present through the execution of the clean_up.csh script as shown previously in Figure 5.22. The CSLM makefile is now ready to be run from the terminal by typing the "make" command in the terminal at the cslm source directory location on the laptop as shown in Figure 5.30.

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/cslm_v1.0/src$ make
```

Fig. 5.30. CSLM BLAS makefile execution command

All of the changes to the run.csh used for the BLAS version of the CSLM algorithm in the previous section are still applicable. The CSLM algorithm is executed as previously done for the BLAS version shown in Figure 5.23. The run.csh file executes in the same manner as that of the BLAS version. Once the OpenBLAS version of the CSLM algorithm completed, the text output from the execution of the run.csh was copied from the terminal and saved in a text file for a comparison later. An example of such output is shown in Figure 5.31 Note that the output is very similar to that of Figure 5.25, in that the algorithm completed in about 22 minutes with an average error of 109.356.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/cslm_v1.0/examples/ngram$ ./run.csh
Build simple back-off LM for comparision
file news09.txt: 2525 sentences, 65595 words, 0 OOVs
0 zeroprobs, logprob= -186780 ppl= 551.986 ppl1= 703.843
Binarizing data
Text to binary converter V1.0, H. Schwenk, LIUM, University of Le Mans, France
 - using word list vocab.txt (14023 words, unk=0, bos=1, eos=2)
 - writing binary representation to file news08.btxt
 - dumping word frequencies to file news08.wlist
 - dumping list of OOV to file news08.oov
 - 2051 lines with 49765 words processed, 8427 uniq words (60.09% of the vocabulary)
 - 0 words were unknown ( 0.00% of the text), -4 new words
Text to binary converter V1.0, H. Schwenk, LIUM, University of Le Mans, France
 - using word list vocab.txt (14023 words, unk=0, bos=1, eos=2)
 - writing binary representation to file news09.btxt
 - dumping word frequencies to file news09.wlist
 - dumping list of OOV to file news09.oov
 - 2525 lines with 65595 words processed, 9726 uniq words (69.36% of the vocabulary)
 - 0 words were unknown ( 0.00% of the text), -4 new words
Sequential machine [3] 3- .. -14024, bs=128, passes=0/0
  Parallel machine 3- .. 768, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
  MachTanh 768-192, bs=128, passes=0/0
  MachSoftmax 192-14024, bs=128, passes=0/0
Opening data description 'train.df'
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Summary of used data:
 - news09.btxt  1.0000 *     63070 =     63070
 - total number of examples: 63070
 - resampling with seed 12345678
 - all resampling coefficients are set to one, loading data once
 - loading all data into memory
 - shuffling data 10 times ... done
Opening data description 'dev.df'
- news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Summary of used data:
 - news09.btxt  1.0000 *     63070 =     63070
 - total number of examples: 63070
 - resampling with seed 12345678
 - all resampling coefficients are set to one, loading data once
 - loading all data into memory
Starting training on host kurt-VPCCA290X pid 6794
 - training on train.df
 - validation on dev.df
 - stopping training at 10 epochs
 - Sequential machine [3] 3- .. -14024, bs=128, passes=0/0
-   Parallel machine 3- .. 768, bs=128, passes=0/0
```

Fig. 5.31. Results of executing the CSLM algorithm on the Sony VPCCA290X laptop using the OpenBLAS libraries

```
-    MachTab 1[14024]-256, bs=128, passes=0/0
 -    MachTab 1[14024]-256, bs=128, passes=0/0
 -    MachTab 1[14024]-256, bs=128, passes=0/0
 -   MachTanh 768-192, bs=128, passes=0/0
 -   MachSoftmax 192-14024, bs=128, passes=0/0
Starting epoch 1 at Mon Feb 29 20:39:28 2016
 - intial lrate=5.0000e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 950.157
 - starting validation ... avrg error: 566.79
Starting epoch 2 at Mon Feb 29 20:41:40 2016
 - intial lrate=4.7598e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 522.23
 - starting validation ... avrg error: 420.934
Starting epoch 3 at Mon Feb 29 20:43:53 2016
 - intial lrate=4.5417e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 407.591
 - starting validation ... avrg error: 332.251
Starting epoch 4 at Mon Feb 29 20:46:07 2016
 - intial lrate=4.3427e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 335.862
 - starting validation ... avrg error: 284.998
Starting epoch 5 at Mon Feb 29 20:48:20 2016
 - intial lrate=4.1603e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 283.561
 - starting validation ... avrg error: 234.436
Starting epoch 6 at Mon Feb 29 20:50:30 2016
 - intial lrate=3.9927e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 242.749
 - starting validation ... avrg error: 199.289
Starting epoch 7 at Mon Feb 29 20:52:41 2016
 - intial lrate=3.8381e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 208.617
 - starting validation ... avrg error: 170.987
Starting epoch 8 at Mon Feb 29 20:54:53 2016
 - intial lrate=3.6950e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 179.748
 - starting validation ... avrg error: 144.58
Starting epoch 9 at Mon Feb 29 20:57:04 2016
 - intial lrate=3.5621e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 155.224
 - starting validation ... avrg error: 125.404
Starting epoch 10 at Mon Feb 29 20:59:18 2016
```

Fig. 5.31. Continued

```
- intial lrate=3.4385e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 134.671
 - starting validation ... avrg error: 109.356
Training stopped
- Sequential machine [3] 3- .. -14024, bs=128, passes=1261400/630700
 -   Parallel machine 3- .. 768, bs=128, passes=1261400/630700
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -   MachTanh 768-192, bs=128, passes=1261400/630700
 -   MachSoftmax 192-14024, bs=128, passes=1261400/630700


Information on machine: example.mach
 - Sequential machine [3] 3- .. -14024, bs=128, passes=1261400/630700
 -   Parallel machine 3- .. 768, bs=128, passes=1261400/630700
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -   MachTanh 768-192, bs=128, passes=1261400/630700
 -   MachSoftmax 192-14024, bs=128, passes=1261400/630700

Evaluating CSLM: example.mach
 - Sequential machine [3] 3- .. -14024, bs=128, passes=1261400/630700
 -   Parallel machine 3- .. 768, bs=128, passes=1261400/630700
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -   MachTanh 768-192, bs=128, passes=1261400/630700
 -   MachSoftmax 192-14024, bs=128, passes=1261400/630700
Using data:
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Perplexity: 109.356
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$
```

Fig. 5.31. Continued

5.8 Performance

Schwenk's CSLM algorithm using the BLAS libraries was run first on the Sony VPCCA290X laptop platform described in Section 5.2 and took 123 min to finish. Thompson et al. [1] reported a similar runtime of the BLAS based version of approximately 160 min on a different laptop platform. The CSLM algorithm was then run using the OpenBLAS libraries on the Sony VPCCA290X laptop and took 22.95 minutes

to finish. The result of executing the CSLM algorithm on three different multicore CPU platforms using BLAS, OpenBLAS, and MKL libraries is summarized in Table 5.3.

Table 5.3
Results of executing the CSLM algorithm on two different multicore CPU platforms using BLAS, OpenBLAS, and MKL libraries

	CPU Platform	Total Time (min)	Epoch Avg. (min)	Avg. Error
BLAS (single core active)	VPCCA290X laptop 2 core Intel i5-2410M processor 2.3 GHz	123	12.38	109.359
OpenBLAS (2 cores active)	VPCCA290X laptop 2 core Intel i5-2410M processor 2.3 GHz	22.95	2.30	109.356
MKL [1]	HP Z800 workstation 12 Intel Xeon x5660 Processors 2.8 GHz	6 [1]	0.6 [1]	109.359 [1]

6. TEGRA K1 IMPLEMENTATION

6.1 Overview

With the baseline CPU performance of the CSLM algorithm on the Sony VPCCA290X laptop captured, attention was turned to the Tegra K1 platform. First, the algorithm was run only on CPU processors of the Tegra K1. Then using CUDA, the computationally intensive portions of the algorithm were ported to run on the Kepler GK20a mobile GPU residing on the Tegra K1. The initial version (v1.0) of Schwenk's open source CSLM toolkit was used as a starting point for the Tegra K1 implementation. It should be noted that only the training and validation portion (cslm_train, line 54 of Figure 5.20) of Schwenk's CSLM toolkit was used for the Tegra K1 implementation. Combining of news08.txt and news09.txt into the text file vocab.txt (lines 19 – 21 of Figure 5.20) and subsequent binarizing of news08.txt and news09.txt (lines 30 – 32 of Figure 5.20) were not performed on the Tegra K1. Rather, the binary files news08.btxt and news09.btxt created in the laptop implementation were transferred over to a directory on the Tegra K1, as described in Sections 6.4 and 6.7 below, to be used in the execution of cslm_train. In addition, the building of a simple 4-gram backoff language model using SRILM (lines 24 – 27 of Figure 5.20) was omitted in the Tegra K1 implementation because it is not used in the execution of cslm_train. Furthermore, the cslm_eval (line 67 of Figure 5.20) and net_info (line 60 of Figure 5.20) commands were not run on the Tegra K1. The rationale is that the execution times used in this study to compare the performance of the algorithm on various platforms is solely based on the starting and ending epoch execution times recorded in the output, such as those shown in Figure 5.25 and Figure 5.31. Note that these execution times originate solely from the cslm_train executable.

As stated in Section 5.5, OpenBLAS was the only available version of BLAS compatible with the ARM A15 multicore CPU processors of the Tegra K1; therefore, the OpenBLAS version was the only CPU based version of the algorithm ported to the Tegra K1.

6.2 Jetson TK1

The Jetson TK1 development platform used in this research was created by NVIDIA to allow developers to explore the capabilities of the Tegra K1. Figure 6.1 shows the Jetson TK1 development platform.

Fig. 6.1. Jetson TK1 Development Platform [25]

The Jetson TK1 provides a standalone platform that contains all the required peripheral components to run the Tegra K1 as well as wide array of interface options for typical applications. Table 6.1 shows a list of the hardware features present on the Jetson TK1 development board.

Table 6.1
Jetson TK1 Hardware Features [26]

Jetson TK1 Hardware
Tega K1 Mobile Processor
Half mini-PCIE slot
Full-size SD/MMC connector
Full-size HDMI port
USB 2.0 port, micro AB
USB 3.0 port, A
RS232 serial port
ALC5639 Realtek Audio Codec with Mic in and Line out RTL8111GS Realtek GigE LAN
SATA data port
2 GB of RAM
16GB of on-board storage

Before any development can occur using the Jetson TK1, some system configuration is required for first time use. With a mouse and keyboard connected to the Jetson TK1 via USB ports, and the Jetson TK1 connected to a HDMI capable display, power is applied, and the Jetson TK1 boots [9]. Once the boot sequence is complete, the user is prompted for login credentials. The default username to be used is “ubuntu”, and the default password is “ubuntu”. Once logged in, the Linux Driver Binary had to be installed using the instructions provided by NVIDIA [9]. With the Linux Driver Binary installed, the next step was to re-flash the Jetson TK1 to install the Linux for Tegra (L4T) operating system onto the Jetson TK1. The Tegra124_Linux_R21.2.0_armhf.tbz2 release of L4T was installed on the Jetson TK1 [9] by folloiwng the instructions from NVIDIA.

6.3 Migration of CLSM Algorithm to Jetson TK1

NVIDIA provides the Jetson TK1 Development Pack (JetPack TK1) as an all-in-one bundle package that installs all the software tools required to develop for the Tegra K1 on the Jetson TK1 platform [27].Version JetPackTK1-1.0-cuda6.5-linux-x64.run of the JetPack TK1 was downloaded to the Sony VPCCA290X laptop from the NVIDIA website and installed using the Custom installation instructions with all of the

components selected as shown in the instructions provided by NVIDIA [28]. These component options included the Jetson TK1 Development Pack, Linux for Tegra R21.1, CUDA Toolkit 6.5 with the “for Ubuntu 12.04 x86_64” and “for L4T” options, GameWorks OpenGL Samples2, Documentation, and Post Setup with the Flash Device, Compile Samples, and Push and Install on Target selected. The resulting CUDA 6.5 Toolkit includes Nsight Development Tools Eclipse Edition version 6.5 for remote debugging and profiling of the Tegra K1 GPU and CPU code simultaneously. NVIDIA Nsight Eclipse Edition is an integrated development environment (IDE) to edit, build, debug, and profile CUDA C applications. It allows development of CUDA applications for either a local (x86_64) system or a remote (x86_64 or ARM) target system [29]. Nsight supports two remote development modes: cross-compilation and synchronbize projects mode. Cross-compiling for ARM on an x86 host requires that all of the ARM libraries to be linked with the application be present on the host system. In synchronize project mode, the source code is synchronized between host and target systems and compiled and linked directly on the remote target [29]. Neither of these remote development modes requires an NVIDIA GPU to be present on the host system [30]. For this work, cross-compilation was used for remote developing CUDA applications for the Jetson TK1. The Sony VPCCA290X laptop served as the host x86_64 platform, and the Jetson TK1 was the ARM target system for the CUDA application. This remote development mode required all of the libraries needed for execution on the ARM processors of the Tegra K1 to be present on the host Sony VPCCA290X laptop.

With the Jetpack host tools installed on the Sony VPCCA290X laptop and the target tools installed on the Jetson TK1, the first step in the migration of the CSLM algorithm over to the Tegra K1 was to create a new Nsight project (cslm_train_tegra_openblas) for the Tegra K1 per online instructions [31]. With the cslm_train_tegra_openblas project created, the next step was to bring all of the required files for the cslm_train_tegra_openblas executable into the Nsight project. Table 6.2 shows the list of files copied from the /src directory of Schwenk’s open source CSLM code version 1.0 into the Nsight project. These include all the files necessary to build an executable equivalent to Schwenk’s cslm_train. The files shown in Table 6.3 are those that are also

in the /src directory of Schwenk's code but that were not included in the Tegra K1 implementation. These files were excluded from the Tegra K1 implementaion since they were only used to generate the text2bin, cslm_eval, and net_info executables, and only the cslm_train (Tegra K1 version was named cslm_train_tegra_openblas) command (line 54 of Figure 5.19) was executed on the Tegra K1. As explained in Section 6.1, the rationale is that the execution times used to compare the performance of the algorithm on various platforms is solely based on the starting and ending epoch execution times recorded in the output, such as those shown in Figure 5.25 and Figure 5.31. Note that these execution times originate solely from the cslm_train executable. Additional files required by cslm_train, such as those residing in the examples/ngram directory of Schwenk's CSLM toolkit, were copied over to the working directory on the Jetson TK1 for the Tegra K1 implementation, as explained in Sections 6.4 and 6.7 below.

Table 6.2
Files From Schwenk's CSLM Toolkit Required to Build the cslm_train Executable and Thus Included in the Nsight project cslm_train_tegra_openblas to be Executed on the Tegra K1

Filename
Blas.h
cslm_train.cpp
Data.cpp
Data.h
DataAscii.cpp
DataAscii.h
DataFile.cpp
DataFile.h
DataNgramBin.cpp
DataNgramBin.h
ErrFct.cpp
ErrFct.h
ErrFctCrossEnt.h
ErrFctCrossEntNgram.h
ErrFctMCE.h
ErrFctMSE.h
ErrFctSoftmCrossEntNgram.cpp
ErrFctSoftmCrossEntNgram.h
Eval.h
EvalNgramBin.cpp
EvalNgramBin.h
Hypo.h
Mach.cpp
Mach.h
MachLin.cpp
MachLin.h
MachMulti.cpp
MachMulti.h
MachPar.cpp
MachPar.h
MachSeq.cpp
MachSeq.h
MachSig.cpp
MachSig.h
MachSoftmax.cpp
MachSoftmax.h
MachTab.cpp
MachTab.h

Table 6.2
Continued

MachTanh.cpp
MachTanh.h
Nbest.h
NbestLM.cpp
NbestLM.h
Tools.cpp
Tools.h
Toolsgz.h
Trainer.cpp
Trainer.h
TrainerNgram.cpp
TrainerNgram.h

Table 6.3
Files From the /src Directory of Schwenk's CSLM Toolkit Excluded from cslm_train_tegra_openblas Nsight Project

Filename
cslm_eval.cpp
ErrFctCrossEnt.cpp
ErrFctCrossEntNgram.cpp
ErrFctMCE.cpp
ErrFctMSE.cpp
Hypo.cpp
MachStacked.cpp
MachStacked.h
MachTabShared.cpp
MachTabShared.h
Nbest.cpp
nbest_cmd.cpp
NbestCSLM.cpp
NbestCSLM.h
NbestLMSRI.cpp
NbestLMSRI.h
net_info.cpp
text2bin.cpp
Toolsgz.cpp

With all of the required source files added to the project, the libraries required for the cslm_train_tegra_openblas project had to be added as well. The CSLM algorithm on the Tegra k1 requires all of the same libraries as those of the laptop version, with the

difference that they must support the ARM A15 architiecture. As mentioned in Section 5.5, the BLAS libraries are not compatible with the ARM A15 architecture of the Tegra K1 so a precompiled version of OpenBLAS, openblas-v0.2.8-armv7-rc2-.tar.gz, that is compatible with the Tegra K1 was downloaded to the Sony VPCCA290X laptop from SourceForge.net [17]. The libopenblas.a, libopenblas.so, libopenblas.so.0, libopenblas_armv7p-r0.2.8.a, and libopenblas_armv7p-r0.2.8.so files from the extraction of this ARM A15 compatible version of OpenBLAS were placed in usr/lib on the laptop, and the symbolic links were added in Nsight to point to the ARM A15 compatible version for the cslm_train_tegra_openblas project. Before these library files could be transferred to /usr/lib on the laptop, the symbolic links used for the laptop implementation had to be removed. A terminal session was opened, and a change directory (cd) command was issued at the terminal to change the directory to /usr/lib as shown in Figure 6.2.

```
kurt@kurt-VPCCA290X:~$ cd ../../usr/lib
```

Fig. 6.2. Terminal commmand to change directory to /usr/lib

The laptop OpenBLAS symbolic links were removed using the terminal commands shown in Figure 6.3.

```
kurt@kurt-VPCCA290X:/usr/lib$ sudo rm libopenblas.so.0
kurt@kurt-VPCCA290X:/usr/lib$ sudo rm libopenblas.so
```

Fig. 6.3. Terminal commands to remove laptop OpenBLAS symbolic link files.

The directory was then changed to the location of the extracted openblas-v0.2.8-armv7-rc2 files as shown in Figure 6.4.

```
kurt@kurt-VPCCA290X:/usr/lib$ cd ../../home/kurt/Desktop/cslm_laptop/
openblas-armv7/lib
```

Fig. 6.4. Directory change to location of the extracted openblas-v0.2.8-armv7-rc2 files.

The files libopenblas.a, libopenblas.so, libopenblas.so.0, libopenblas_armv7p-r0.2.8.a, and libopenblas_armv7p-r0.2.8.so were then copied to the usr/lib directory using the terminal commands as shown in Figure 6.5.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/openblas-armv7/lib$ sudo cp
*.* /usr/lib
[sudo] password for kurt:
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/openblas-armv7/lib$
```

Fig. 6.5. Terminal commands to copy libopenblas.a, libopenblas.so, libopenblas.so.0, libopenblas_armv7p-r0.2.8.a, and libopenblas_armv7p-r0.2.8.so to usr/lib on the Sony VPCCA290X laptop

As in the laptop implementation, the SRILM libraries are required with the cslm_train_tegra_openblas project build for the Tegra K1 because the CSLM toolkit makes use of the oolm.a (-loolm), dstruct.a (-ldstruct), misc.a (-lmisc), z.a (-lz), and m.a (-lm) libraries generated by the SRILM build as shown in Appendix A.1 line 20, but version 1.6.0 that was built for use in the laptop implementation is not compatible with the ARM A15 architecture. To generate a version of the SRILM 1.6.0 libraries that support the ARM A15 architecture, a modification to the build process descibed in Section 5.6 was required. In the /common sub-directory of the srilm-1.6.0 directory there is a file Makefile.machine.i686-ubuntu-32 that is used for the i686 32-bit build of the SRILM libraries for the Ubuntu operating system. The makefile.machine.i686-ubuntu-32 shown in Figure 6.6 makes use of the GCC compiler present in Ubuntu to compile the libraries for the i686 architecture.

```
#
#    File:   Makefile.machine.i686-ubuntu-32
#    Author: The SRI DECIPHER (TM) System
#    Date:   Tue Jan 25 13:49:15 PST 2011
#
#    Description:
#     Machine dependent compilation options and variable definitions
#     for Ubuntu Linux/i686 platform, forcing 32bit build on 64bit
systems
#
#    Copyright (c) 1999-2011 SRI International.  All Rights Reserved.
#
#    $Header: /home/srilm/CVS/srilm/common/Makefile.machine.i686-
ubuntu-32,v 1.1 2011/01/25 21:49:52 stolcke Exp $
#

   # Use the GNU C compiler.
   GCC_PATH = /usr/bin/
   GCC_FLAGS =  -m32 -Wall -Wno-unused-variable -Wno-uninitialized
   CC = $(GCC_PATH)gcc $(GCC_FLAGS)
   CXX = $(GCC_PATH)g++ $(GCC_FLAGS) -DINSTANTIATE_TEMPLATES

   # Optional compilation flags.
   OPTIMIZE_FLAGS = -g -O3
   DEBUG_FLAGS = -g -DDEBUG
   PROFILE_FLAGS = -g -pg -O3

   # Optional linking flags.
   EXPORT_LDFLAGS = -s

   # Shared compilation flags.
   CFLAGS = -D_FILE_OFFSET_BITS=64 $(ADDITIONAL_CFLAGS) $(INCLUDES)
   CXXFLAGS = -D_FILE_OFFSET_BITS=64 $(ADDITIONAL_CXXFLAGS) $(INCLUDES)

   # Shared linking flags.
   LDFLAGS = $(ADDITIONAL_LDFLAGS) -L$(SRILM_LIBDIR)

   # Other useful compilation flags.
   ADDITIONAL_CFLAGS =
   ADDITIONAL_CXXFLAGS =

   # Other useful include directories.
   ADDITIONAL_INCLUDES =

   # Other useful linking flags.
   ADDITIONAL_LDFLAGS =

   # Other useful libraries.
   ADDITIONAL_LIBRARIES = -lm -ldl

   # run-time linker path flag
   RLD_FLAG = -R
```

Fig. 6.6. Laptop SRILM-1.6.0 Makefile.machine.i686-ubuntu-32.

```
# No Tcl support by default (32bit libraries generally not present)
NO_TCL = X
TCL_LIBRARY =

# No ranlib
RANLIB = :

# Generate dependencies from source files.
GEN_DEP = $(CC) $(CFLAGS) -MM

GEN_DEP.cc = $(CXX) $(CXXFLAGS) -MM

# Run lint.
LINT = lint
LINT_FLAGS = -DDEBUG $(CFLAGS)

# Location of gawk binary
GAWK = /usr/bin/awk

# Location of perl binary
PERL = /usr/bin/perl
```

Fig. 6.6. Continued

This file was copied and renamed Makefile.machine.arm-linux so it could be modified to make use of the NVIDIA nvcc compiler for the Tegra K1 ARM A15 architecture. Figure 6.7 shows the Makefile.machine.arm-linux makefile with the modifications to use the NVIDIA nvcc. Note that the modifications are denoted in red font in Figure 6.7

```
#
#    File:   Makefile.machine.i686-ubuntu-32
#    Author: The SRI DECIPHER (TM) System
#    Date:   Tue Jan 25 13:49:15 PST 2011
#
#    Description:
#     Machine dependent compilation options and variable definitions
#     for Ubuntu Linux/i686 platform, forcing 32bit build on 64bit
systems
#
#    Copyright (c) 1999-2011 SRI International.  All Rights Reserved.
#
#    $Header: /home/srilm/CVS/srilm/common/Makefile.machine.i686-
ubuntu-32,v 1.1 2011/01/25 21:49:52 stolcke Exp $
#

   # Use the GNU C compiler.
   GCC_PATH = /usr/local/cuda-6.5/bin/nvcc
   GCC_FLAGS =  -m32 -ccbin arm-linux-gnueabihf-g++-4.6  --target-cpu-
architecture ARM
   CC = $(GCC_PATH) $(GCC_FLAGS)
   CXX = $(GCC_PATH) $(GCC_FLAGS) -DINSTANTIATE_TEMPLATES

   # Optional compilation flags.
   OPTIMIZE_FLAGS = -g -O0
   DEBUG_FLAGS = -g -DDEBUG
   PROFILE_FLAGS = -g -pg -O0

   # Optional linking flags.
   EXPORT_LDFLAGS = -s

   # Shared compilation flags.
   CFLAGS = -D_FILE_OFFSET_BITS=64 $(ADDITIONAL_CFLAGS) $(INCLUDES)
   CXXFLAGS = -D_FILE_OFFSET_BITS=64 $(ADDITIONAL_CXXFLAGS) $(INCLUDES)

   # Shared linking flags.
   LDFLAGS = $(ADDITIONAL_LDFLAGS) -L$(SRILM_LIBDIR)

   # Other useful compilation flags.
   ADDITIONAL_CFLAGS =
   ADDITIONAL_CXXFLAGS =

   # Other useful include directories.
   ADDITIONAL_INCLUDES =

   # Other useful linking flags.
   ADDITIONAL_LDFLAGS =

   # Other useful libraries.
   ADDITIONAL_LIBRARIES = -lm -ldl

   # run-time linker path flag
   RLD_FLAG = -R
```

Fig. 6.7. Tegra K1 SRILM-1.6.0 Makefile.machine.arm-linux

```
# No Tcl support by default (32bit libraries generally not present)
   NO_TCL = X
   TCL_LIBRARY =

   # No ranlib
   RANLIB = :

   # Generate dependencies from source files.
   GEN_DEP = $(CC) $(CFLAGS) -MM

   GEN_DEP.cc = $(CXX) $(CXXFLAGS) -MM

   # Run lint.
   LINT = lint
   LINT_FLAGS = -DDEBUG $(CFLAGS)

   # Location of gawk binary
   GAWK = /usr/bin/awk

   # Location of perl binary
   PERL = /usr/bin/perl
```

Fig. 6.7. Continued

To compile the Tegra K1 ARM A15 compatible SRILM libraries on the Sony VPCCA290X laptop, navigate at the terminal to the directory containing the makefile, as shown in Figure 6.8, and execute the command shown in Figure 6.9.

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/srilm-1.6.0$
```

Fig. 6.8. Directory location of the SRILM makefile

```
kurt@kurt-VPCCA290X:/home/kurt/cslm_laptop/srilm-1.6.0$make MACHINE_TYPE=arm-linux World
```

Fig. 6.9. SRILM makefile command to generate ARM A15 version for Tegra K1

With the build of the ARM A15 compatible version of the SRILM-1.6.0 libraries complete, the libraries and their search paths were updated in Nsight. The libraries dstruct, oolm, misc, m, and openblas were added as shown in Figure 6.10. The SRILM library path, /home/kurt/Desktop/cslm_laptop/srilm-1.6.0/lib/arm-linux, was added to the Nsight project as shown in Figure 6.10 to include the ARM A15 compatible version of the SRILM libraries. The addition of the /usr/lib Library search path in Figure. 6.10

enabled the Nsight project to access the OpenBLAS symbolic link file locations and the OpenBLAS ARM A15 compatible libraries.

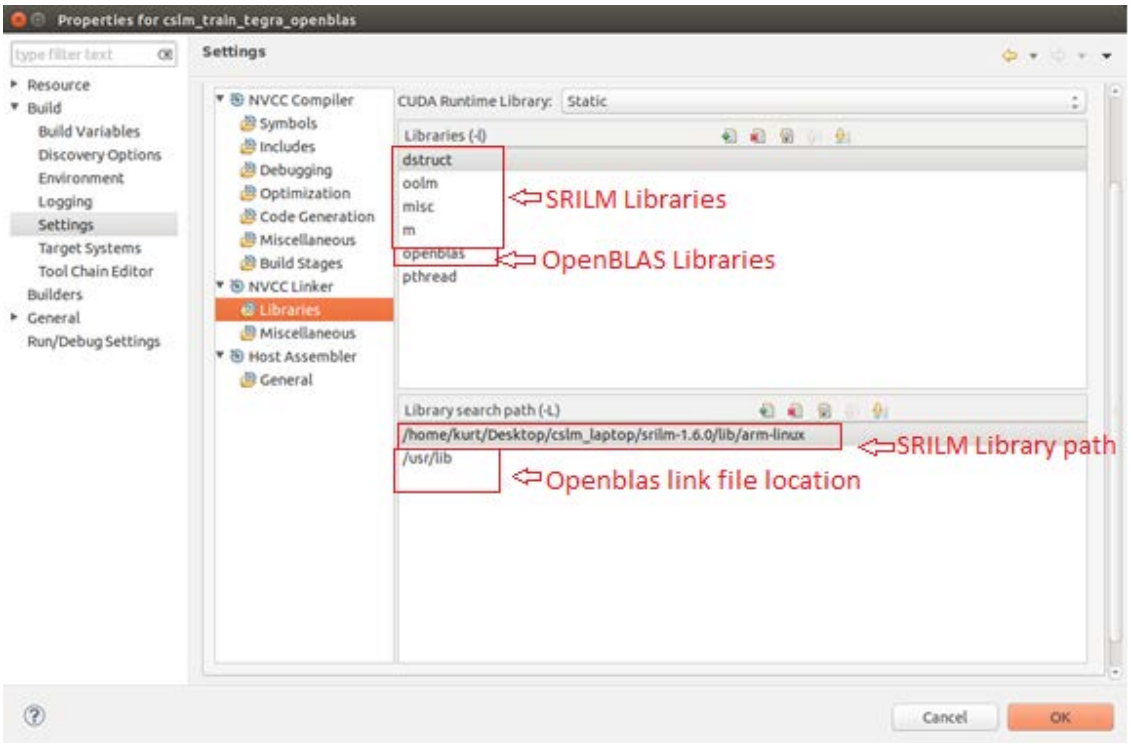

Fig 6.10. OpenBLAS and SRILM library path updates in Nsight.

The cslm_train_tegra_openblas Nsight project was updated as shown in Figure 6.11 to include the location of the include directories, /home/kurt/Desktop/cslm_laptop/openblas-armv7/include, containing the extracted ARM A15 compatible version of the openblas-v0.2.8-armv7-rc2.

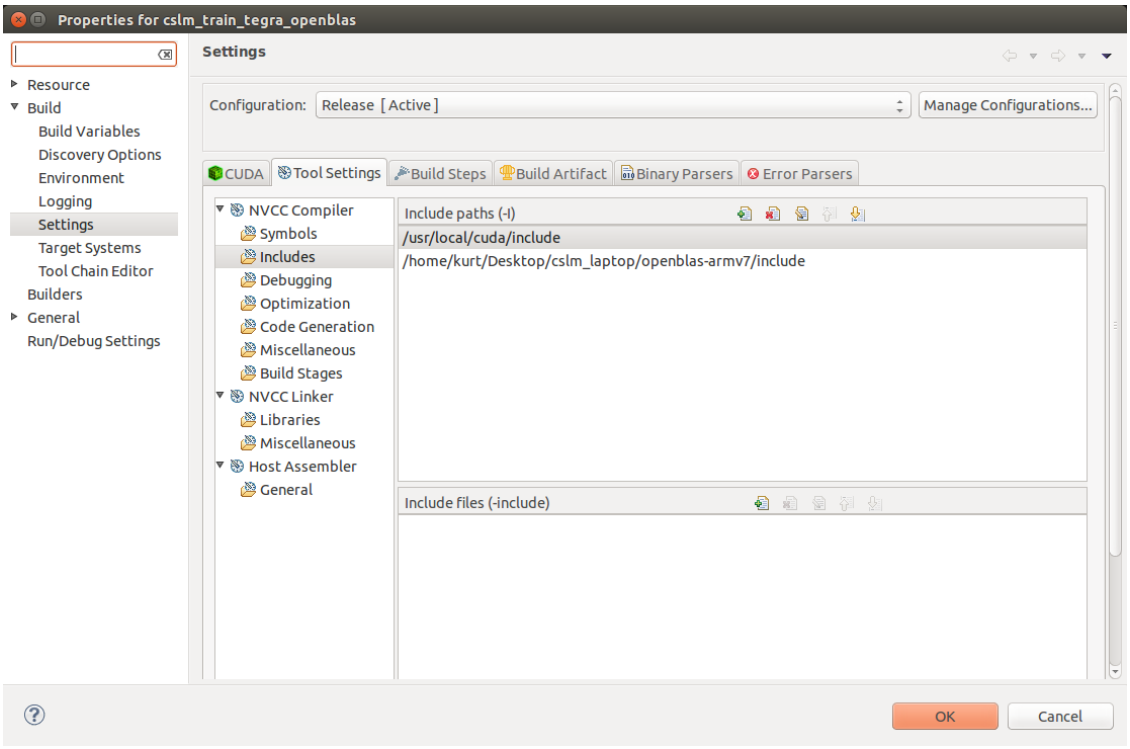

Fig. 6.11. Nsight Include directories for SRILM and OpenBLAS for ARM A15

With the required modifications made to the source files to implement the OpenBLAS version of the CSLM algorithm on the Tegra K1, an executable had to be generated from Nsight. From within the cslm_train_tegra_openblas project in Nsight, “Build Project” is selected as shown in Figure 6.12.

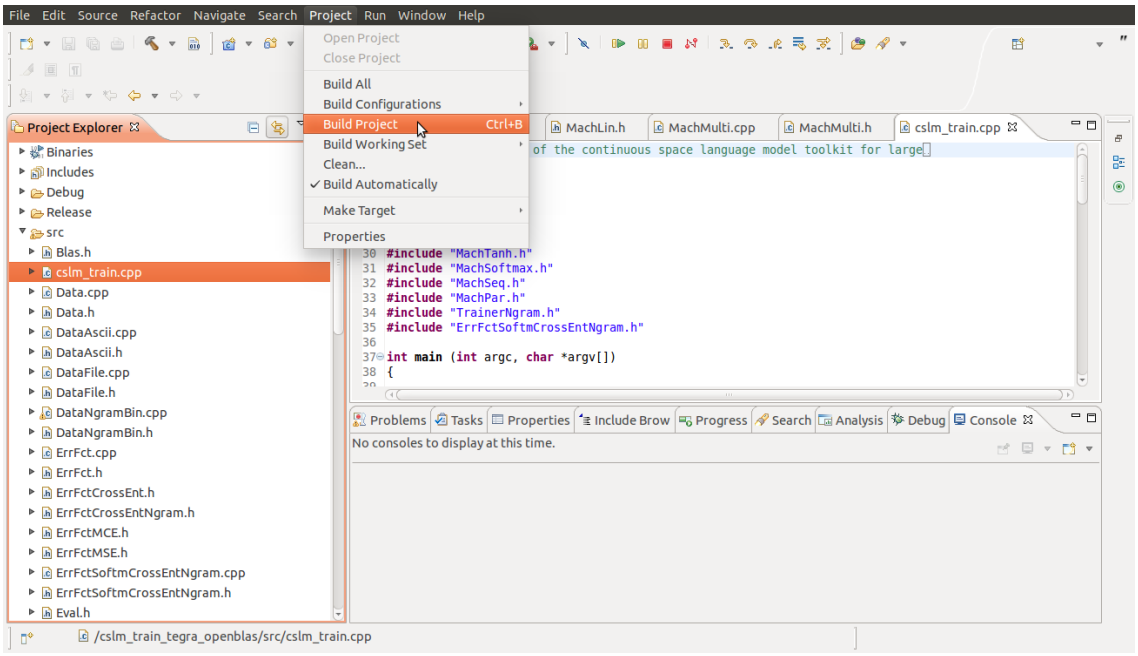

Fig. 6.12. cslm_train_tegra_openblas Build Project from Nsight

When the build of the cslm_train_tegra_openblas completes, the “Console” tab from within Nsight states “Finished Building Target : cslm_train_tegra_openblas” as shown Figure 6.13. The cslm_train_tegra_openblas executable was now ready to be run on the Tegra K1.

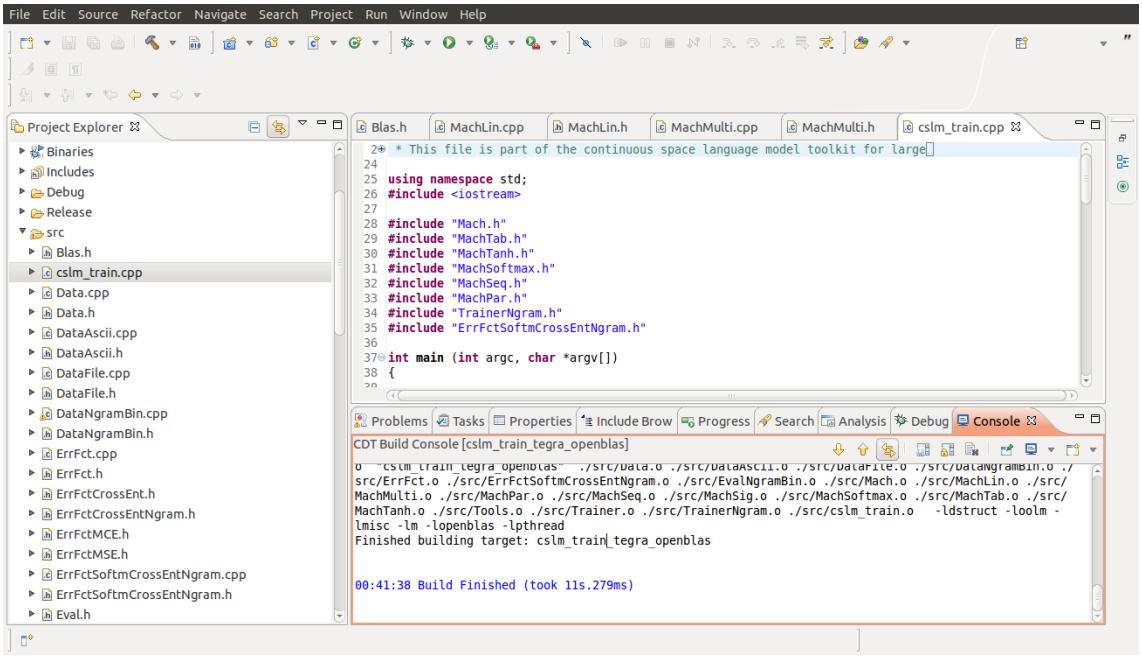

Fig. 6.13. cslm_train_tegra_openblas Build Project Complete

6.4 Tegra K1 OpenBLAS execution

While the remote development for the Tegra K1 was accomplished using NVIDIA Nsight Eclipse Edition with cross compilation mode per online instructions [29], the remote execution as described there was not performed. Rather, the cslm_train_tegra_openblas executable was created on the VPCCA290X laptop as described in Section 6.3 and then transferred over to the Tegra K1, as described later in this Section. This was done to ensure that any execution time due to Nsight was not included in the recorded execution times of the algorithm. In addition to transferring to the Tegra K1 the project executable that was created on the laptop using Nsight, additional files required by cslm_train were also copied over to the Tegra K1. The files that were transferred and the procedures used to do so are explained in detail in this section. This section also describes the procedure used to remotely execute the Tegra K1 project from the Sony VPCCA290X laptop.

Before the cslm_train_tegra_openblas executable can be executed on the Tegra K1, the libopenblas.a, libopenblas.so, libopenblas.so.0, libopenblas_armv7p-r0.2.8.a, and

libopenblas_armv7p-r0.2.8.so files from the extraction of openblas-v0.2.8-armv7-rc2-.tar.gz had to be placed in usr/lib on the Tegra K1. With Jetson TK1 connected to a router, the executable could be copied over and run remotely. A terminal session was launched by using the shortcut “Ctrl+Alt+T” from the keyboard, and the directory was changed to the location of the extracted openblas-v0.2.8-armv7-rc2-.tar.gz files as shown in Figure 6.14.

```
kurt@kurt-VPCCA290X:~$ cd /Desktop/cslm_laptop/openblas-armv7/lib
```

Fig. 6.14. Change directory terminal command to change to extracted openblas-v0.2.8-armv7-rc2-.tar.gz files

The files were then remotely copied over to the Tegra K1’s Desktop using the Secure Copy (scp) command as shown in Figure 6.15.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/openblas-armv7/lib$ scp *.*
ubuntu@10.0.0.2:/home/ubuntu/Desktop/
libopenblas.a                                 100%   14MB   1.4MB/s
00:10
libopenblas_armv7p-r0.2.8.a                   100%   14MB   1.6MB/s
00:09
libopenblas_armv7p-r0.2.8.so                  100% 8556KB   2.1MB/s
00:04
libopenblas.so                                100% 8556KB   1.7MB/s
00:05
libopenblas.so.0                              100% 8556KB   1.7MB/s
00:05
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/openblas-armv7/lib$
```

Fig. 6.15. Remote transfer terminal command to transfer extracted openblas-v0.2.8-armv7-rc2-.tar.gz files

With the files successfully transferred over to the Tegra K1, a remote session on the Tegra K1 had to be established to move the transferred files to the usr/lib directory on the Tegra K1. Figure 6.16 shows the terminal commands to establish the remote session with the Tegra K1.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/openblas-armv7/lib$ ssh
ubuntu@10.0.0.2
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.10.40-ged4f697 armv7l)

 * Documentation:  https://help.ubuntu.com/

Last login: Sat Jan  1 00:00:23 2000
ubuntu@tegra-ubuntu:~$
```

Fig. 6.16. Terminal Command for remote session.

The files were then transferred from the desktop of the Tegra K1 to the /usr/lib directory with the terminal command shown in Figure 6.17. Note that the terminal prompt in Figure 6.17 has changed from the laptop's kurt@kurt-VPCCA290X: to the Tegra K1's ubuntu@tegra-ubuntu:. All terminal commands from this point forward are with respect to the Tegra K1.

```
ubuntu@tegra-ubuntu:~/Desktop$ sudo cp libopenblas.a /usr/lib
[sudo] password for ubuntu:
ubuntu@tegra-ubuntu:~/Desktop$ sudo cp libopenblas.so /usr/lib
ubuntu@tegra-ubuntu:~/Desktop$ sudo cp libopenblas.so.0 /usr/lib
ubuntu@tegra-ubuntu:~/Desktop$ sudo cp libopenblas_armv7p-r0.2.8.a
/usr/lib
ubuntu@tegra-ubuntu:~/Desktop$ sudo cp libopenblas_armv7p-r0.2.8.so
/usr/lib
ubuntu@tegra-ubuntu:~/Desktop$
```

Fig. 6.17. Remote terminal command to transfer files from the desktop of the Tegra K1 to the to /usr/lib directory of the Tegra K1

As with the laptop implementation of OpenBLAS, the Tegra K1 implementation also requires the libgfortran libraries. Using Synaptic Package Manager on the Jetson TK1, libgfortran3 was installed. This installed the files libgfortran.so.3 and libgfortran.so.3.0.0 at /usr/arm-linux-gnueabhf/lib on the Jetson TK1.

With the required libraries transferred over to the Tegra K1, the cslm_train_tegra_openblas executable had to be copied over the the Jetson TK1 before it could be executed. To do this from the laptop a terminal session was launched by using the shortcut "Ctrl+Alt+T" from the keyboard. From the terminal the directory was changed to the directory containing the cslm_train_tegra_openblas executable using the command shown in Figure 6.18.

```
kurt@kurt-VPCCA290X:~$ cd cuda-workspace/cslm_train_tegra_openblas/Release/
```

Fig. 6.18. Directory change to cslm_train_tegra_openblas executable

A cslm_test directory was remotely created on the Jetson TK1 using the Make Directory (mkdir) command shown in Figure 6.19.

```
kurt@kurt-VPCCA290X:~/cuda-workspace/cslm_train_tegra_openblas/Release$ssh ubuntu@10.0.0.2
mkdir /home/ubuntu/Desktop/cslm_test
```

Fig. 6.19. Command to create cslm_test directory on the desktop of Jetson TK1 remotely

The cslm_train_tegra_openblas executable was then transferred to the Jetson TK1 using the command shown in Figure 6.20.

```
kurt@kurt-VPCCA290X:~/cuda-workspace/cslm_train_tegra_openblas/Release$ scp
cslm_train_tegra_openblas ubuntu@10.0.0.2:/home/ubuntu/Desktop/ cslm_test
```

Fig. 6.20. Copy command to copy over the cslm_train_tegra_openblas executable to Jetson TK1

When the transfer of the cslm_train_tegra_openblas executable to the Tegra K1 is complete, the terminal will display the transfer progress as shown in Figure 6.21.

```
cslm_train_tegra_openblas                     100% 3928KB   3.8MB/s   00:01
kurt@kurt-VPCCA290X:~/cuda-workspace/cslm_train_tegra_openblas/Release$
```

Fig. 6.21. Successful transfer of the cslm_train_tegra_openblas executable to the Jetson TK1

Next the files required for use with the cslm_train_tegra_openblas executable had to be transferred over to the Jetson TK1 as well. From the terminal, navigate to the cslm example directory on the laptop as shown in Figure 6.22.

```
kurt@kurt-VPCCA290X:~/cuda-workspace/ cslm_train_tegra_openblas/Release$ cd
../../../Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram/
```

Fig. 6.22. Directory change to location of required files for cslm on Jetson TK1

From the /examples/ngram directory of Schwenk's open source CSLM code, the training data file, train.df, and the validation data file, dev.df, were transferred from the Sony VPCCA290X laptop to the Jetson TK1 by using the command from the terminal prompt as shown in Figure 6.23.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/cslm_v1.0/examples/ngram$ scp *.df
ubuntu@10.0.0.2:/home/ubuntu/Desktop/gpu_cslm_test
```

Fig. 6.23. Command for train.df and dev.df transfer to Jetson TK1

Figure 6.24 shows the successful transfer of the dev.df and train.df files to the Jetson TK1. The next files required to be transferred over to the Jetson TK1 was the example.mach file in the /examples/ngram directory of Schwenk's CSLM code.

```
dev.df                        100%  239    0.2KB/s   00:00
train.df                      100%  568    0.6KB/s   00:00
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptopcslm_v1.0/examples/ngram$
```

Fig. 6.24. Successful transfer of train.df and dev.df to Jetson TK1

The command from the terminal prompt was executed as shown in Figure 6.25 to transfer the example.mach file to the Jetson TK1.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop /cslm_v1.0/examples/ngram$ scp example.mach
ubuntu@10.0.0.2:/home/ubuntu/Desktop/cslm_test
```

Fig. 6.25. Command to copy example.mach to Jetson TK1

Figure 6.26 shows the successful transfer of the example.mach file to the Jetson TK1. The news08.btxt and news09.bxt files also had to be transferred over to the Jetson TK1.

```
example.mach                  100%   25MB   2.1MB/s   00:12
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/cslm_v1.0/examples/ngram$
```

Fig. 6.26. Successful transfer of example.mach to Jetson TK1

Figure 6.27 shows the command from the terminal prompt used to transfer the news08.btxt and news09.btxt files to the Jetson TK1.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop/cslm_v1.0/examples/ngram$ scp *.btxt
ubuntu@10.0.0.2:/home/ubuntu/Desktop/cslm_test
```

Fig. 6.27. Command to copy news08.btxt and news09.btxt to Jetson TK1

Figure 6.28 shows the successful transfer of the news08.btxt and news09.btxt to the Jetson TK1. With all of the required files to run the OpenBLAS version of the CSLM algorithm transferred over to the Jetson TK1, it was then necessary to remote into the Jetson TK1 from the laptop in order to execute the cslm_train_tegra_openblas executable.

```
news08.btxt                   100%  210KB 210.5KB/s   00:00
news09.btxt                   100%  276KB 276.0KB/s   00:00
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$
```

Fig. 6.28. Successful transfer of news08.btxt and news09.btxt to Jetson TK1

Figure 6.29 shows the command from the terminal prompt on the laptop used to set up the remote connection to the Jetson TK1.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$ ssh
ubuntu@10.0.0.2
```

Fig. 6.29. Command to remote into the Jetson TK1

Figure 6.30 shows the successful remote connection to the Jetson TK1. Note that the terminal prompt in Figure 6.30 has changed from the laptop's kurt@kurt-VPCCA290X: to the Jetson's ubuntu@tegra-ubuntu:. All terminal commands from this point forward are with respect to the Jetson TK1.

```
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.10.40-ged4f697 armv7l)

 * Documentation:  https://help.ubuntu.com/

Last login: Fri Jan 14 01:37:02 2000 from 10.0.0.3
ubuntu@tegra-ubuntu:~$
```

Fig. 6.30. Successful connection to Jetson TK1

Since the Tegra K1 is designed for mobile use, it contains power reduction systems to control when parts of the hardware should run faster or slower, or be turned off, based on runtime use. The automatic turning on or off of the 4 main CPU cores and the 5th companion core is mostly done in the L4T kernel using "cpuquiet", which provides a means of dynamically hot-plugging CPU cores based upon workload/policy [32]. For this work, it was desired to disable the cpuquiet feature and to specify the use of two CPU cores. Since the OpenBLAS version of the CSLM algorithm on the laptop used two CPU cores, the OpenBLAS version on the Tegra K1 was allowed the usage of two CPU cores to execute the CLSM algorithm on the Jetson TK1. To do this, the command prompt from the terminal was used to navigate to the tegra_cpuquiet directory as shown in Figure 6.31 [32].

```
ubuntu@tegra-ubuntu:cd ../../sys/devices/cpu/cpuquiet/tegra_cpuquiet
```

Fig. 6.31. Changing directory to tegra_cpu quiet

The command at the terminal prompt in Figure 6.32 calls sudo with the /bin/bash command to switch to a root session with the root privleges required to disable cpu_quiet and enable the second CPU core.

```
ubuntu@tegra-ubuntu:/sys/devices/system/cpu/cpuquiet/tegra_cpuquiet$ sudo /bin/bash
[sudo] password for ubuntu:
root@tegra-ubuntu:/sys/devices/system/cpu/cpuquiet/tegra_cpuquiet#
```

Fig. 6.32. Getting root privileges so can disable cpu quiet

Figure 6.33 shows the command used at the terminal prompt to disable cpu_quiet.

```
root@tegra-ubuntu:/sys/devices/system/cpu/cpuquiet/tegra_cpuquiet# echo 0 > enable
root@tegra-ubuntu:/sys/devices/system/cpu/cpuquiet/tegra_cpuquiet#
```

Fig. 6.33. Disable the cpu_quiet

With cpu_quiet disabled, the command shown if Figure 6.34 was then used at the terminal prompt to change the directory to the cpu1 directory.

```
root@tegra-ubuntu:/sys/devices/system/cpu# cd ../../cpu1
```

Fig. 6.34. Change to cpu1 directory

The command shown in Figure 6.35 was then used at the terminal prompt to manually enable the second CPU core within the Tegra K1.

```
root@tegra-ubuntu:/sys/devices/system/cpu/cpu1# echo 1 > online
root@tegra-ubuntu:/sys/devices/system/cpu/cpu1#
```

Fig. 6.35. Allow cpu1 to come on

With the second CPU core enabled, the directory was then changed to the location of the cslm_train_tegra_openblas executable as shown in Figure 6.36.

```
root@tegra-ubuntu:/sys/devices/system/cpu/cpu1# cd ../../../../../home/ubuntu/Desktop/cslm_test/
```

Fig. 6.36. Change to cslm_test directory

The cslm_train_tegra_openblas executable was then launched from the remote terminal session on the Tegra K1 by executing the terminal command as shown in Figure 6.37. As explained in Section 6.1, the ./run.csh file that was used in the laptop implementation is not used here since the focus is only the training and validation portion of the CSLM algorithm.

```
ubuntu@tegra-ubuntu:~/Desktop/cslm_test$ ./ cslm_train_tegra_openblas train.df dev.df example.mach
256 192 14024 4 128 5e-3 4e-7 3e-5 10
```

Fig. 6.37. Execution of the OpenBLAS cslm_train_tegra_openblas executable on the Jetson TK1

Once the execution of the cslm_train_tegra_openblas completes, the text output from the execution of the output from the terminal is copied and saved in a text file for a comparison later. An example of the output is shown in Figure 6.38. The training and validation portion of the algorithm took about 83 minutes with an average error of 109.356.

```
 ubuntu@tegra-ubuntu:~/Desktop/cslm_test$ ./cslm_train_tegra_openblas train.df dev.df example.mach
256 192 14024 4 128 5e-3 4e-7 3e-5 10
Sequential machine [3] 3- .. -14024, bs=128, passes=0/0
  Parallel machine 3- .. 768, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
  MachTanh 768-192, bs=128, passes=0/0
  MachSoftmax 192-14024, bs=128, passes=0/0
Opening data description 'train.df'
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Summary of used data:
 - news09.btxt  1.0000 *     63070 =     63070
 - total number of examples: 63070
 - resampling with seed 12345678
 - all resampling coefficients are set to one, loading data once
 - loading all data into memory
 - shuffling data 10 times ... done
Opening data description 'dev.df'
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Summary of used data:
 - news09.btxt  1.0000 *     63070 =     63070
 - total number of examples: 63070
 - resampling with seed 12345678
 - all resampling coefficients are set to one, loading data once
 - loading all data into memory
Starting training on host tegra-ubuntu pid 1925
 - training on train.df
 - validation on dev.df
 - stopping training at 10 epochs
 - Sequential machine [3] 3- .. -14024, bs=128, passes=0/0
 -   Parallel machine 3- .. 768, bs=128, passes=0/0
 -     MachTab 1[14024]-256, bs=128, passes=0/0
 -     MachTab 1[14024]-256, bs=128, passes=0/0
 -     MachTab 1[14024]-256, bs=128, passes=0/0
 -   MachTanh 768-192, bs=128, passes=0/0
 -   MachSoftmax 192-14024, bs=128, passes=0/0
Starting epoch 1 at Sat Jan 29 02:14:03 2000
 - intial lrate=5.0000e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 950.157
 - starting validation ... avrg error: 566.79
Starting epoch 2 at Sat Jan 29 02:22:32 2000
 - intial lrate=4.7598e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 522.23
 - starting validation ... avrg error: 420.934
Starting epoch 3 at Sat Jan 29 02:31:01 2000
 - intial lrate=4.5417e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
```

Fig. 6.38. OpenBLAS Tegra K1 (CPU) Execution Results

```
- epoch finished, 63070 examples seen, average error: 407.591
 - starting validation ... avrg error: 332.251
Starting epoch 4 at Sat Jan 29 02:39:30 2000
 - intial lrate=4.3427e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 335.862
 - starting validation ... avrg error: 284.998
Starting epoch 5 at Sat Jan 29 02:47:59 2000
 - intial lrate=4.1603e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 283.561
 - starting validation ... avrg error: 234.436
Starting epoch 6 at Sat Jan 29 02:56:28 2000
 - intial lrate=3.9927e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 242.749
 - starting validation ... avrg error: 199.289
Starting epoch 7 at Sat Jan 29 03:04:56 2000
 - intial lrate=3.8381e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 208.617
 - starting validation ... avrg error: 170.987
Starting epoch 8 at Sat Jan 29 03:13:25 2000
 - intial lrate=3.6950e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 179.748
 - starting validation ... avrg error: 144.58
Starting epoch 9 at Sat Jan 29 03:21:53 2000
 - intial lrate=3.5621e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 155.224
 - starting validation ... avrg error: 125.404
Starting epoch 10 at Sat Jan 29 03:30:22 2000
 - intial lrate=3.4385e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 134.671
 - starting validation ... avrg error: 109.356
Training stopped
 - Sequential machine [3] 3- .. -14024, bs=128, passes=1261400/630700
 -   Parallel machine 3- .. 768, bs=128, passes=1261400/630700
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -   MachTanh 768-192, bs=128, passes=1261400/630700
 -   MachSoftmax 192-14024, bs=128, passes=1261400/630700
ubuntu@tegra-ubuntu:~/Desktop/cslm_test$
```

Fig. 6.38. Continued

6.5 Tegra K1 GPU implementation

As explained in Section 5.1, The CSLM algorithm as originally implemented by Schwenk is a CPU version that provides the option of incorporating high performance freely available BLAS or ATLAS libraries to accomplish fast matrix multiplications. Alternatively, it provides the option of utilizing the Intel MKL libraries, for a small fee. Versions v2, v3, and v4 of Schwenk's CSLM code incorporate support for various desktop GPUs, but as of the latest version (version 4 June 2015), the GPU on the Tegra K1 (mobile processor platform) is not supported. Schwenk's GPU versions of the CSLM use the NVIDIA cuBLAS Library [33], which is a GPU version of BLAS. It also incorporates functions from the NVIDIA Performance Primitive Library (NPP) [34], a highly optimized set of libraries for performing CUDA accelerated processing. In addition, Schwenk includes some CUDA kernel functions [11] in his GPU implementations. In creating a GPU implementation for use on the Tegra K1, some of his function calls were modified and/or incorporated, as explained in the following subsections.

Starting with CUDA 6.0, the cuBLAS Library has two sets of Application Programmer Interfaces (API), the original cuBLAS and CUBLASXT [33]. The original cuBLAS interface requires allocation of matrices and vectors on the GPU, initialization of these by uploading of data from the CPU to the GPU, calling the desired cuBLAS function, and then transferring results from the GPU back to the CPU. However, in using the CUBLASXT API, data is allocated on the host, and the Library handles dispatching of the operation to one or more GPUs, at the request of the user. For the research reported here, the original cuBLAS API was utilized.

As explained in Section 3.3., in general, when a kernel is called from the host, the input data is copied over the interface bus from the CPU to the GPU memory. In the case of devices that support Unified Memory (devices that support CUDA 6.0 and higher), explicit memory copies are no longer required [11]. The underlying system manages memory copies in a transparent manner, hidden from the user. In architectures such as the Tegra K1 in which the CPU and the GPU are integrated on the same die and have access

to shared memory, data transfer overhead could potentially be eliminated or reduced. For the research reported here, the function *cudaMallocManaged* was invoked to create and access data from CUDA Unified Memory, thus eliminating the need for explicit memory copies between host and device. The *cudaMallocManaged* function returns a pointer valid from both host and device code, allowing direct access of GPU data from the host.

With the OpenBLAS implementation of the CSLM algorithm on the Tegra K1 complete, the computationally intensive portions of the algorithm were then ported over to the GPU in the Tegra K1. These computationally intensive portions pertained to those forward pass operations given in Equations 2.4 - 2.7 from Sections 2.3 and 2.4 as well as the backward pass operations described by Thompson et al. [1]. To do this, a project named cslm_train_cuda_rev2 was created in Nsight to incorporate the Tegra K1 GPU implementation. The OpenBLAS Tegra K1 CPU code had to be modified within Nsight to make use of the CUDA cuBLAS library, CUDA NPP library, and CUDA kernel functions [33] [34] [11]. This required multiple changes to the files from Schwenk's version 1.0 of the CSLM algorithm [15]. Schwenk's CSLM versions v2, v3, and v4 software releases [15] support the use of GPUs, so the modifications to the CSLM GPU implementation on the Tegra K1 were based on the functions Schwenk ported over to the GPU in CSLM v2 and v3 releases. Table 6.4 lists the files that were modified from CSLM version 1.0 in order to port the computationally intensive portions of the algorithm over to the GPU in the Tegra K1 and Table 6.5 lists the files used from Schwenk's CSLM versions v2 and v3 releases in the Tegra K1 GPU implementation. The following subsections detail the changes made to each of the files from Table 6.4 and Table 6.5. In addition, note that the original Schwenk code for each of these files is listed in Appendix B; the modified versions for use in the Tegra K1 GPU implementation are provided in Appendix D.

Table 6.4
Files From Schwenk's CSLM version 1.0 Toolkit That Were Modified in Creating a GPU Version for Execution on the Tegra K1

File Name
ErrFct.cpp
ErrFct.h
ErrFctSoftmCrossEntNgram.cpp
EvalNgramBin.cpp
Mach.cpp
Mach.h
MachLin.cpp
MachPar.cpp
MachSeq.cpp
MachSig.cpp
MachSoftmax.cpp
MachTab.cpp
MachTanh.cpp
MachTanh.h
NbestCSLM.h
Trainer.cpp
TrainerNgram.cpp

Table 6.5
Files From Schwenk's CSLM version 2.0 and version 3.0 Toolkits That Were Modified in Creating a GPU Version for Execution on the Tegra K1

File Name
GPU.cu
GPU.cuh

6.5.1 GPU.cuh

Appendix D.4 shows Schwenk's original Gpu.cuh file for the CSLM version 3.0 release, and Appendix D.5 shows the CSLM version 2.0 release. Both the 2.0 release and the 3.0 release were used as the source for the GPU implementation on the Tegra K1 [15]. The combined sections used in the Tegra K1 GPU implementation Gpu.cuh file are shown in Appendix D.6.The comments shown in red in Appendix D.6 show which function prototype statements in the GPU.cuh file for the Tegra K1 implementation were used from Schwenk's verison 2 and version 3 releases.

6.5.2 GPU.cu

Appendix D.1 shows Schwenk's original Gpu.cu file for the CSLM version 3.0 release, and Appendix D.2 shows the CSLM version 2.0 release listed in Table 6.5. Both the 2.0 release and the 3.0 release were used as the source for the GPU implementation on the Tegra K1 [15]. The combined sections used in the Tegra K1 GPU implementation Gpu.cu file are shown in Appendix D.3. The GPU.cu file for the Tegra K1 GPU implementation shown in Appendix D.3 shows the *GpuMachTabForw*, *GpuMachTabBackw*, and *GpuErrFctSoftmCrossEntNgramCalcGrad* function and kernel definitions from Schwenk's verison 2 release. Appendix D.3 also shows the use of Schwenk's version 2 function and kernel definition of *GpuErrFctSoftmCrossEntNgramCalcValue* where there is the addition of the CUDA Runtime API *cudaFree* call. This is used to deallocate memory on the GPU that has been previously allocated and is no longer used [11]. The functions used from Schwenk's

version 3 release shown in Appendix D.3 were functions that were either not present in the version 2 release or in the case of the *GpuMachSoftmaxForw* function, were an enhancement over the version 2 *GpuMachSoftmaxForw*. The *GpuCopyVectorToMatrix* function and kernel definition provided a vector to matrix copy method that was not present in the version 2 release. The *GpuBatchedAXPY* function and kernel definitions is a batched GPU version that Schwenk found to provide better performance than the cuBLAS based AXPY function present in the version 2 release.

The *GpuMachSoftmaxForw* function from Schwenk's version 3 release was modified to ignore the warpsize check since this implementation is meant only for the Tegra K1 with a warpsize of 32. The int *n_blocks* assignment was changed from 1024 to the 2048 maximum allowable number of threads per block of the Tegra K1. The *error* reporting function call from Schwenk's version 3 release within the *GpuMachSoftmaxForw* function definition was not used. The *error* function call was commented out for the Tegra K1 implementation since the changes to the tools.h file from the Schwenk's version 3 release were not adopted in the Tegra K1 implementation.

6.5.3 my_cuda.h

The my_cuda.h header file shown in Figure 6.39 was not in Schwenk's code but was created for use in the GPU implementation on the Tegra K1 as a general purpose header file that could be included to whatever files that needed access to the components therein [33]. This header file was included in several of the revised files, such as MachLin.cpp, MachSoftmax.cpp, MachTanh.cpp, and TrainerNgram.cpp. The cublas_v2.h is part of the cuBLAS library and with its inclusion cuBLAS library APIs can be used. The npps.h provides access the NPP library APIs [34]. The cuda_runtime.h and cuda_runtime_api.h provide access to the CUDA Runtime APIs [11], which provides functions the ability to enable, allocate, and deallocate memory on the GPU. The *cublasHandle_t cublas_handle* and *cublasStatus_t cublas_stat* are part of the cuBLAS library [33]. The *cublas_stat* is used to report errors returned by cuBLAS function calls and *cublas_handle* is used to allocate the resources on the Host and GPU prior to making cuBLAS function calls [33].

```
#ifndef _my_cuda_h
#define _my_cuda_h

#include "cublas_v2.h" //cublas header file
#include "cuda_runtime.h"
#include "npps.h"
#include "cuda_runtime_api.h"

extern cublasHandle_t cublas_handle;
extern cublasStatus_t cublas_stat;

#endif
```

Fig. 6.39. my_cuda.h

6.5.4 ErrFct.cpp

Figure 6.40 shows the original include preprocessor directives for ErrFct.cpp. For the modified version of ErrFct.cpp used in the GPU implementation on the Tegra K1, an additional preprocessor directive was added to include the cuda_runtime.h header file as shown in Figure 6.41 [11].

```
#include "Tools.h"
#include "ErrFct.h"
```

Fig. 6.40. Original include preprocessor directives in ErrFct.cpp

The cuda_runtime.h header file is part of the CUDA Application Programming Interface (API) and allows the host access to functions to handle device management, context management, memory management, code module management, execution control, texture reference management, and interoperability with OpenGL and Direct3D [11] [35]. Host files that make use of CUDA Runtime API to access the functions to perform the host control and management operations require the inclusion of the cuda_runtime.h file [11]. In order to allocate or deallocate memory on the GPU, specific CUDA allocation and deallocation functions must be called. These memory allocation and deallocation functions are part of the CUDA Runtime API [11].

```
#include "Tools.h"
#include "ErrFct.h"
#include "cuda_runtime.h"
```

Fig. 6.41. Modified include preprocessor directives in ErrFct.cpp to include cuda_runtime.h

Figure 6.42 shows the original *ErrFct::ErrFct(Mach &mach)* constructor initialization within ErrFct.cpp where the *grad* pointer (highlighted in red in Figure 6.42) is allocated on the host memory as an array of size *dim*bsize*.

```
ErrFct::ErrFct (Mach &mach)
 : dim(mach.GetOdim()), bsize(mach.GetBsize()),
   output(mach.GetDataOut()), target(NULL), grad(new REAL[dim*bsize])
{
//cerr << "Constructor ErrFct: alloc gradient of size " << dim << endl;
}
```

Fig. 6.42. Original ErrFct::ErrFct(Mach &mach) in ErrFct.cpp

Figure 6.43 shows the modification of the *ErrFct::ErrFct(Mach &mach)* constructor within ErrFct.cpp to incorporate Unified Memory by including *the cudaMallocManaged* function call to allocate the *grad* array as managed memory, allowing *grad* to be used by both the host and the device during execution [11].

```
ErrFct::ErrFct (Mach &mach)
 : dim(mach.GetOdim()), bsize(mach.GetBsize()),
   output(mach.GetDataOut()), target(NULL)
{
cudaMallocManaged(&grad, dim*bsize*sizeof(REAL));
//cerr << "Constructor ErrFct: alloc gradient of size " << dim << endl;
}
```

Fig. 6.43. Modified ErrFct::ErrFct()within ErrFct.cpp

Figure 6.44 shows the addition within ErrFct.cpp of the class deconstructor *ErrFct::~ErrFct()* which uses the *cudaFree* deallocation function to free the memory used by *grad* [11].The addition of the class deconstructor was required because the deconstruction for *grad* was performed in the class definition in ErrFct.h. With the use of managed memory, the deallocation of the memory needed to be performed using the

cudaFree call that is included in the CUDA Runtime API. Therefore the class deconstructor was moved to ErrFct.cpp.

```
ErrFct::~ErrFct()
{
 cudaFree(grad);
}
```

Fig. 6.44. Addition of ErrFct::~ErrFct() within ErrFct.cpp

6.5.5 ErrFct.h

The original ErrFct class definition within ErrFct.h is shown in Figure 6.45 where the deallocation of the memory for array *grad* was originally performed.

```
class ErrFct
{
private:
protected:
 int       dim;                    // output dimension of machine
 int       bsize;
 REAL  *output;            // pointer to output data (stored in machine)
 REAL  *target;            // pointer to target data (stored in trainer)
 REAL  *grad;                      // calculated gradient (stored in this class)
public:
 ErrFct(Mach&);
 virtual ~ErrFct() { delete [] grad; }
 void SetOutput(REAL *p_output) {output=p_output; }
 void SetTarget(REAL *p_target) {target=p_target; }
 REAL *GetGrad() {return grad; };
 virtual REAL CalcValue(int=0);          // Calculate value of error function
 virtual REAL CalcGrad(int=0);                  // calculate NEGATIF gradient of error function
};
```

Fig. 6.45. Original ErrFct class declaration within ErrFct.h

Figure 6.46 shows the modification to the ErrFct class definition within ErrFctSoftmCrossEntNgram.cpp for the GPU implementation where the deallocation for *grad* has been removed leaving only class deconstructor *ErrFct::~ErrFct()* since the deallocation is now performed in the class decontstructer *ErrFct::~ErrFct()* located within ErrFct.cpp using *cudaFree* as shown in Figure 6.44 [11].

```
class ErrFct
{
private:
protected:
 int       dim;                          // output dimension of machine
 int       bsize;
 REAL  *output;              // pointer to output data (stored in machine)
 REAL  *target;              // pointer to target data (stored in trainer)
 REAL  *grad;                          // calculated gradient (stored in this class)
public:
 ErrFct(Mach&);
 virtual ~ErrFct();
 void SetOutput(REAL *p_output) {output=p_output; }
 void SetTarget(REAL *p_target) {target=p_target; }
 REAL *GetGrad() {return grad; };
 virtual REAL CalcValue(int=0);                  // Calculate value of error function
 virtual REAL CalcGrad(int=0);                           // calculate NEGATIF gradient of error function
};
```

Fig. 6.46. Modified ErrFct class declaration within ErrFct.h

6.5.6 ErrFctSoftmCrossEntNgram.cpp

Figure 6.47 shows the original include preprocessor directives for ErrFctSoftmCrossEntNgram.cpp.The header include of ErrFctSoftmCrossEntNgram.cpp for the Tegra K1 GPU implementation of the cslm_train_cuda_rev2 executable required the addition of the cuda_runtime.h header file as well as the Gpu.cuh header file as shown in Figure 6.48.

```
#include "Tools.h"
#include "ErrFctSoftmCrossEntNgram.h"
```

Fig. 6.47. Original include preprocessor directives within ErrFctSoftmCrossEntNgram.cpp

The Gpu.cuh header file contains the GPU kernel function declarations so that kernel function calls can be made from within ErrFctSoftmCrossEntNgram.cpp.

```
#include "Tools.h"
#include "ErrFctSoftmCrossEntNgram.h"
#include "Gpu.cuh"
#include "cuda_runtime.h"
```

Fig. 6.48. Modified include preprocessor directives within ErrFctSoftmCrossEntNgram.cpp

Figure 6.49 shows the original *CalcValue(int eff_bsize)* function definition in ErrFctSoftmCrossEntNgram.cpp where the value *err* is being calculated sequentially within a *for* loop.

```
REAL ErrFctSoftmCrossEntNgram::CalcValue(int eff_bsize) {
 REAL *optr=output;
 REAL *tptr=target;
 double err=0.0;

 if (eff_bsize<=0) eff_bsize=bsize;
 for (int b=0; b<eff_bsize; b++) {
  if (*tptr<0 || *tptr>=dim) {
    printf("ErrFctSoftmCrossEntNgram::CalcValue(): target out of bounds (%d) must be in
    [0,%d[\n",(uint)*tptr,dim);
    Error();
  }
//printf("b=%d, tidx=%f, out=%f\n", b, *tptr, optr[(uint) *tptr]);
   err += log(optr[(uint) *tptr++]);
//printf("err=%f\n",err);
   optr += dim;
 }
 return (REAL) err; // TODO: normalize ?
}
```

Fig. 6.49. Original ErrFctSoftmCrossEntNgram::CalcValue(int eff_bsize) within ErrFctSoftmCrossEntNgram.cpp

Figure 6.50 shows the modification to function *ErrFctSoftmCrossEntNgram::CalcValue(int eff_bsize)*, within ErrFctSoftmCrossEntNgram.cpp, where the *for* loop has been removed, and the value for *err* is returned by the *GpuErrFctSoftmCrossEntNgramCalcValue(eff_bsize, dim, output, target)* function call, which is defined in the revised GPU.cu header file shown in Appendix D3. The calculation for *err* is has now been ported over to the GPU within the Tegra K1.

```
REAL ErrFctSoftmCrossEntNgram::CalcValue(int eff_bsize) {
 REAL *optr=output;
 REAL *tptr=target;
 double err=0.0;

 if (eff_bsize<=0) eff_bsize=bsize;
//printf("b=%d, tidx=%f, out=%f\n", b, *tptr, optr[(uint) *tptr]);
//printf("err=%f\n",err);
 err = GpuErrFctSoftmCrossEntNgramCalcValue(eff_bsize, dim, output, target);
 return (REAL) err; // TODO: normalize ?
}
```

Fig. 6.50. Modified ErrFctSoftmCrossEntNgram::CalcValue(int eff_bsize) within ErrFctSoftmCrossEntNgram.cpp

Figure 6.51 shows the original function definition in ErrFctSoftmCrossEntNgram.cpp for *ErrFctSoftmCrossEntNgram::CalcGrad(int eff_bsize)* where the *err* value is calculated on the CPU within a nested *for* loop.

```
REAL ErrFctSoftmCrossEntNgram::CalcGrad(int eff_bsize) {
 REAL *optr=output;
 REAL *tptr=target;
 REAL *gptr=grad;
 REAL err=0.0;
 uint      tidx;

 if (eff_bsize<=0) eff_bsize=bsize;
 for (int b=0; b<eff_bsize; b++) {
  for (int i=0; i<dim; i++) gptr[i] = -optr[i];
  tidx=(uint) *tptr++;
  if (tidx<0 || tidx>=(uint) dim) {
   printf("ErrFctSoftmCrossEntNgram::CalcGrad(): target out of bounds (%d) must be in
[0,%d[\n",tidx,dim);
    Error();
  }
  err += log(optr[tidx]);
  gptr[tidx] += 1.0;
  gptr+=dim; optr+=dim;
 }
 return (REAL) err; // TODO: normalize ?
}
```

Fig. 6.51. Original ErrFctSoftmCrossEntNgram::CalcGrad(int eff_bsize) within ErrFctSoftmCrossEntNgram.cpp

Figure 6.52 shows the modification to the *ErrFctSoftmCrossEntNgram::CalcGrad(int eff_bsize)* function definition in ErrFctSoftmCrossEntNgram.cpp where the variable *err* is now the value returned by the *GpuErrFctSoftmCrossEntNgramCalcGrad* function call, which is defined in the revised GPU.cu header file shown in Appendix D3. This function call ports the calculation to the Tegra K1 GPU.

```
REAL ErrFctSoftmCrossEntNgram::CalcGrad(int eff_bsize) {
 REAL err=0.0;
 uint      tidx;
 if (eff_bsize<=0) eff_bsize=bsize;
 err=GpuErrFctSoftmCrossEntNgramCalcGrad(eff_bsize, dim, output, grad, target);
 return (REAL) err; // TODO: normalize ?
}
```

Fig. 6.52. Modified ErrFctSoftmCrossEntNgram::CalcGrad(int eff_bsize) within ErrFctSoftmCrossEntNgram.cpp

6.5.7 EvalNgramBin.cpp

The original include preprocessor directives for the EvalNgramBin.cpp file are shown in Figure 6.53.

```
#include "Tools.h"
#include "EvalNgramBin.h"

#include <algorithm>
```

Fig. 6.53. Original include preprocessor directives within EvalNgramBin.cpp

Figure 6.54 shows the modified include preprocessor directives for the revised version of EvalNgramBin.cpp used in the GPU implementation. As discussed in Section 6.5.4, the inclusion of the cuda_runtime.h allows the host file to make CUDA Runtime API function calls.

```
#include "Tools.h"
#include "EvalNgramBin.h"

#include <algorithm>
#include "cuda_runtime.h"
```

Fig. 6.54. Modified include preprocessor directives within EvalNgramBin.cpp

Figure 6.55 shows the original *EvalNgramBin::EvalNgramBin(Mach &p_mach, uint p_max_req)* class definition in EvalNgramBin.cpp where the *buf_input* variable is allocated in memory as an array of size *idim*bsize*.

```
EvalNgramBin::EvalNgramBin(Mach &p_mach, uint p_max_req)
 : mach(p_mach), max_req(p_max_req)
{

 idim=mach.GetIdim(); odim=mach.GetOdim(); bsize=mach.GetBsize();

 if (odim < 16) {
  fprintf(stderr,"EvalNgramBin: output dimension of the machine is suspiciously small (%d)\n", odim);
  Error();
 }

 buf_input = new REAL[idim*bsize];
 mach.SetDataIn(buf_input);
}
```

Fig. 6.55. Original EvalNgramBin::EvalNgramBin(Mach &p_mach, uint p_max_req) defined in EvalNgramBin.cpp

Figure 6.56 shows the modifed version of the class definition *EvalNgramBin ::EvalNgramBin(Mach &p_mach, uint p_max_req)* in EvalNgramBin.cpp in which the memory allocation has been replaced with the *cudaMallocManaged* memory allocation for *buf_input* for the GPU implementation.

```
EvalNgramBin::EvalNgramBin(Mach &p_mach, uint p_max_req)
 : mach(p_mach), max_req(p_max_req)
{

 idim=mach.GetIdim(); odim=mach.GetOdim(); bsize=mach.GetBsize();

 if (odim < 16) {
  fprintf(stderr,"EvalNgramBin: output dimension of the machine is suspiciously small (%d)\n", odim);
  Error();
 }

 cudaMallocManaged(&buf_input, idim*bsize*sizeof(REAL));
 mach.SetDataIn(buf_input);
}
```

Fig. 6.56. Modified EvalNgramBin::EvalNgramBin(Mach &p_mach, uint p_max_req) defined in EvalNgramBin.cpp

Figure 6.57 shows the original class deconstructor *EvalNgramBin::~EvalNgramBin()* within EvalNgramBin.cpp that deallocates the memory allocated for the *buf_input* variable.

```
EvalNgramBin::~EvalNgramBin()
{
 for (vector<NgramReq*>::iterator it=req.begin(); it<req.end(); ++it) delete *it;
 delete [] buf_input;
}
```

Fig. 6.57. Original EvalNgramBin::~EvalNgramBin() in EvalNgramBin.cpp

Figure 6.58 shows the modified *EvalNgramBin::~EvalNgramBin()* class deconstructor for the GPU implementation. It uses *cudaFree* to deallocate *buf_input*

```
EvalNgramBin::~EvalNgramBin()
{
 for (vector<NgramReq*>::iterator it=req.begin(); it<req.end(); ++it) delete *it;
 cudaFree(buf_input);
}
```

Fig. 6.58. Modified EvalNgramBin::~EvalNgramBin() in EvalNgramBin.cpp

6.5.8 Mach.cpp

Figure 6.59 shows the original include preprocessor directives for Schwenk's Mach.cpp. Figure 6.60 shows the modified include preprocessor directives for the GPU implementation. The cuda_runtime.h allows the host file to make CUDA Runtime API function calls as discussed in Section 6.5.4.

```
using namespace std;
#include <iostream>

#include "Tools.h"
#include "Mach.h"
#include "MachTab.h"
#include "MachTabShared.h"
#include "MachLin.h"
#include "MachSig.h"
#include "MachTanh.h"
#include "MachSoftmax.h"
#include "MachSeq.h"
#include "MachPar.h"
```

Fig. 6.59. Original include preprocessor directives of Mach.cpp

The npps.h include preprocessor directives allows the usage of the NVIDIA Performace Primitives (NPP) Library to perform various CUDA accelerated functions and operations [34].

```
using namespace std;
#include <iostream>

#include "Tools.h"
#include "Mach.h"
#include "MachTab.h"
#include "MachTabShared.h"
#include "MachLin.h"
#include "MachSig.h"
#include "MachTanh.h"
#include "MachSoftmax.h"
#include "MachSeq.h"
#include "MachPar.h"
#include "cuda_runtime.h"
#include "npps.h"
```

Fig. 6.60. Modified include preprocessor directives of Mach.cpp

Figure 6.61 shows the original *Mach::do_alloc()* memory allocation function definition in Mach.cpp for the *data_out*, *data_in*, *grad_in*, and *grad_out* variables.

```
void Mach::do_alloc()
{
 if (odim*bsize>0) {
   data_out=::new REAL[odim*bsize];
   if (!data_out) Error ("can't allocate memory for data_out");
 }
 else data_out=NULL;
 data_in=NULL; // (luint) this) should be set later by SetDataIn()
 if (idim*bsize>0) {
   grad_in=::new REAL[idim*bsize];
   if (!grad_in) Error ("can't allocate memory for grad_in");
 }
 else grad_in=NULL;
 grad_out=NULL; // (luint) this) should be set later by SetGradOut()
}
```

Fig. 6.61. Original Mach::do_alloc() in Mach.cpp

Figure 6.62 show the modification to *Mach::do_alloc()* that replaces the memory allocations with the *cudaMallocManaged* memory allocations for the GPU implementation.

```
void Mach::do_alloc()
{
 if (odim*bsize>0) {
  cudaMallocManaged(&data_out, odim*bsize*sizeof(REAL));
  if (!data_out) Error ("can't allocate memory for data_out");
 }
 else data_out=NULL;
 cudaMallocManaged(&data_in, sizeof(REAL));
 data_in=NULL; // (luint) this) should be set later by SetDataIn()
 if (idim*bsize>0) {
  cudaMallocManaged(&grad_in, idim*bsize*sizeof(REAL));
  if (!grad_in) Error ("can't allocate memory for grad_in");
 }
 else grad_in=NULL;
 cudaMallocManaged(&grad_out, sizeof(REAL));
 grad_out=NULL; // (luint) this) should be set later by SetGradOut()
}
```

Fig. 6.62. Modified Mach::do_alloc() in Mach.cpp

Figure 6.63 shows the original memory class decontructor *Mach::~Mach()*, defined in Mach.cpp, for the *data_out* and *grad_in* variables.

```
Mach::~Mach()
{
 if (data_out) delete [] data_out;
 if (grad_in) delete [] grad_in;
}
```

Fig. 6.63. Original Mach::~Mach() in Mach.cpp

Figure 6.64 shows the modified memory class deconstructor *Mach::~Mach()* for the GPU implementation where the memory deallocation for the variables *data_out* and *grad_in* was changed to use *cudaFree*.

```
Mach::~Mach()
{
 cudaFree(data_out);
 cudaFree(grad_in);
}
```

Fig. 6.64. Modified Mach::~Mach() in Mach.cpp

Figure 6.65 shows the original *Mach::Forw(int eff_bsize)* function definition in Mach.cpp. Figure 6.66 shows the modified version of *Mach::Forw(int eff_bsize)* for the GPU implementation where the *memcpy* operation that copies *data_in* to *data_out* has been replaced with the *nppsCopy_32f* function, to enable the operation to be performed on the GPU.

```
void Mach::Forw(int eff_bsize)
{
 if (!data_in)
   Error("Mach::Forw(): input data is not set");
 if (idim!=odim)
   Error("Mach::Forw(): call to default Forw() function with different dimensions");
 if (eff_bsize<=0) eff_bsize=bsize;
 memcpy(data_out, data_in, eff_bsize*idim*sizeof(REAL));
 nb_forw += eff_bsize;
}
```

Fig. 6.65. Original Mach::Forw(int eff_bsize) in Mach.cpp

With the migration of portions of the algorithm to the GPU, some data operations such as the *memcopy* operation in Figure 6.65 are no longer performed on the host device within the Tegra K1. Figure 6.66 shows the *memcpy* function replaced by the *nppsCopy_32f* function from the NPP library [34]. The *nppsCopy_32f* function is used here to perform a vector copy of the *data_out* to the *data_in* from within the device. The use of the *nppsCopy_32f* function removes the need for any memory transfers or updates between the host and device, thereby keeping the data transfer overhead between the host and device to a minimum [35].

```
void Mach::Forw(int eff_bsize)
{
 if (!data_in)
   Error("Mach::Forw(): input data is not set");
 if (idim!=odim)
   Error("Mach::Forw(): call to default Forw() function with different dimensions");
 if (eff_bsize<=0) eff_bsize=bsize;
 nppsCopy_32f(data_in, data_out, eff_bsize*idim);
 nb_forw += eff_bsize;
}
```

Fig. 6.66. Modified Mach::Forw(int eff_bsize) in Mach.cpp

Figure 6.67 shows the original function definition in Mach.cpp for *Mach::Backw (const float lrate, const float wdecay, int eff_bsize)* where the *memcpy* operation on the CPU host is copying the *grad_out* data to *grad_in*.

```
void Mach::Backw (const float lrate, const float wdecay, int eff_bsize)
{
 if (!grad_out)
   Error("Mach::Backw(): output gradient is not set");
 if (idim!=odim)
   Error("Mach::Backw(): call to default Train() function with different dimensions");
 if (eff_bsize<=0) eff_bsize=bsize;
 memcpy(grad_in, grad_out, eff_bsize*idim*sizeof(REAL));
 nb_backw += eff_bsize;
}
```

Fig. 6.67. Original Mach::Backw (const float lrate, const float wdecay, int eff_bsize) in Mach.cpp

Figure 6.68 shows the modified *Mach::Backw (const float lrate, const float wdecay, int eff_bsize)* function definition in Mach.cpp for the GPU implementation that makes use to the *nppsCopy_32f* function to perform the copy of the *grad_out* data to *grad_in* on the GPU.

```
void Mach::Backw (const float lrate, const float wdecay, int eff_bsize)
{
 if (!grad_out)
   Error("Mach::Backw(): output gradient is not set");
 if (idim!=odim)
   Error("Mach::Backw(): call to default Train() function with different dimensions");
 if (eff_bsize<=0) eff_bsize=bsize;
 nppsCopy_32f(grad_out, grad_in, eff_bsize*idim);
 nb_backw += eff_bsize;
}
```

Fig. 6.68. Modified Mach::Backw (const float lrate, const float wdecay, int eff_bsize) in Mach.cpp

6.5.9 MachLin.cpp

Figure 6.69 shows the original include preprocessor directives for Schwenk's MachLin.cpp. Figure 6.70 shows the modified version of the include preprocessor directives for the GPU implementation.

```
using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
#include "MachLin.h"
#include "Blas.h"
```

Fig. 6.69. Original include preprocessor directives for MachLin.cpp

The orignal include preprocessor directives were modified for the GPU implementation to include the my_cuda.h and Gpu.cuh header files, as shown in Figure 6.70. As stated in Section 6.5.6, the Gpu.cuh header file contains the GPU kernel function declarations. The inclusion of this header file in MachLin.cpp enables these kernel function calls to be made from within MachLin.cpp for the GPU implementation.

```
using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();
#include "my_cuda.h"
#include "Tools.h"
#include "MachLin.h"
#include "Blas.h"
#include "Gpu.cuh"
```

Fig. 6.70. Modified include preprocessor directives for MachLin.cpp

Figure 6.71 shows the original *MachLin::MachLin(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int p_nbbw)* constructor initialization where the memory allocations for the variables *b* and *w* are performed using the C++ *new* operator, as shown in red in Figure. 6.71.

```
MachLin::MachLin(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int
p_nbbw)
 : Mach(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
  if (odim>0) {
    b = new REAL[odim];
    if (!b) Error ("can't allocate memory for bias of linear machine");
  }
  else b=NULL;
  if (idim*odim>0) {
    w = new REAL[idim*odim];
    if (!w) Error ("can't allocate memory for weights of linear machine");
  }
  else w=NULL;
}
```

Fig. 6.71. Original MachLin::MachLin(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int p_nbbw) in MachLin.cpp

These memory allocations for *w* and *b* of Figure 6.71 were modified as shown in red in Figure 6.72 to use the *cudaMallocManaged* function for the GPU implementation, enabling the use of unified memory.

```
MachLin::MachLin(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int
p_nbbw)
 : Mach(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
  if (odim>0) {
    cudaMallocManaged(&b, odim*sizeof(REAL));
    if (!b) Error ("can't allocate memory for bias of linear machine");
  }
  else b=NULL;
  if (idim*odim>0) {
            cudaMallocManaged(&w, idim*odim*sizeof(REAL));
            if (!w) Error ("can't allocate memory for weights of linear machine");
  }
  else w=NULL;
}
```

Fig. 6.72. Modified MachLin::MachLin(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int p_nbbw) in MachLin.cpp

Figure 6.73 shows the original class deconstructor *MachLin::~MachLin()* where the memory deallocation of variables *w* and *b* was performed using the C++ *delete* operator.

```
MachLin::~MachLin()
{

#if 0
 printf("W:\n");
 for (int od=0;od<odim;od++) {
  for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
  printf("\n");
 }
 printf("b: ");
 for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
 printf("\n");
#endif
 if (b) delete [] b;
 if (w) delete [] w;
}
```

Fig. 6.73. Original MachLin::~MachLin() in MachLin.cpp

Figure 6.74 show the modified version of the class deconstructor *MachLin::~MachLin()* for the GPU implementation, in which the memory deallocations of *w* and *b* of Figure 6.73 were replaced by *cudaFree*.

```
MachLin::~MachLin()
{

#if 0
 printf("W:\n");
 for (int od=0;od<odim;od++) {
  for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
  printf("\n");
 }
 printf("b: ");
 for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
 printf("\n");
#endif
 cudaFree(b);
 cudaFree(w);
}
```

Fig. 6.74. Modified MachLin::~MachLin() in MachLin.cpp

Figure 6.75 shows the original function definition in MachLin.cpp for *MachLin::Forw(int eff_bsize)* . The matrix multiplication operations in the forward pass described by Thompson et al. [1] are performed using the BLAS/OpenBLAS *gemm*

function calls [1] in the original Schwenk (v1) code. Figure 6.76 shows the modifications made to the *MachLin::Forw(int eff_bsize)* function definition for the GPU implementation.

```
void MachLin::Forw(int eff_bsize)
{
 if (!data_in)
   Error("MachLin::Forw(): input data is not set");

 if (eff_bsize<=0) eff_bsize=bsize;

#if 0
 printf("Forw %p, bsize=%d\n", (void*)this, eff_bsize);
 printf("W: %dx%d\n",odim,idim);
 for (int od=0;od<odim;od++) {
  for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
  printf("\n");
 }
 printf("b:\n");
 for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
 printf("\n");
#endif
#if 0
for (int e=0; e<eff_bsize; e++) {
 printf("B %d inp:", e);
 for (int i=0; i<idim; i++) printf(" %7.5f", data_in[i+e*idim]);
 printf("\n");
}
#endif

#ifdef BLAS
 if (eff_bsize>1) { // BLAS block mode: GEMM
  int e,o;
  REAL *optr, *bptr;

   // copy bias <eff_bsize> times into result matrix
  for (e=0, optr=data_out; e<eff_bsize; e++) {
   for (o=0, bptr=b; o<odim; o++) *optr++ = *bptr++;
  }
  call_gemm (data_out, w, data_in, 1.0, odim, eff_bsize, idim);
 }
 else { // BLAS vector mode: GEMV
  call_gemv (data_out, w, data_in, b, odim, idim);
 }
#else
 for (int e=0; e<eff_bsize; e++) {
    // simple matrix vector multiply, TODO: verify bsize
    // TODO: W is stored in BLAS (Fortan) format: colum major !!
  //cout << "forw ex " << e << endl;
  REAL *wptr=w;
  for (int o=0; o<odim; o++) {
   REAL s=b[o];
   for (int i=0; i<idim; i++) s+=wptr[i*odim+o]*data_in[i+e*idim];
```

Fig. 6.75. Original MachLin::Forw(int eff_bsize) in MachLin.cpp

```
    data_out[o+e*odim]=s;
   }
  }
#endif
  nb_forw += eff_bsize;

#if 0
for (int e=0; e<eff_bsize; e++) {
  printf("B %d out:", e);
  for (int i=0; i<odim; i++) printf(" %7.5f", data_out[i+e*odim]);
  printf("\n\n");
}
#endif

}
```

Fig. 6.75. Continued

Figure 6.76 shows the replacement of the nested for loops used to copy the bias vector into the result matrix with the *GpuCopyVectorToMatrix* GPU function to perform the equivalent operation on the GPU. The *GpuCopyVectorToMatrix* function prototype statement is included in the Gpu.cuh header file (Appendix D.6) that was added for the GPU implementation. The *GpuCopyVectorToMatrix* function definition and associated kernel function are defined in the Gpu.cu file (Appendix D.3) described in Section 6.5.2.

```
void MachLin::Forw(int eff_bsize)
{
 if (!data_in)
   Error("MachLin::Forw(): input data is not set");

 if (eff_bsize<=0) eff_bsize=bsize;

#if 0
 printf("Forw %p, bsize=%d\n", (void*)this, eff_bsize);
 printf("W: %dx%d\n",odim,idim);
 for (int od=0;od<odim;od++) {
  for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
  printf("\n");
 }
 printf("b:\n");
 for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
 printf("\n");
#endif
#if 0
for (int e=0; e<eff_bsize; e++) {
 printf("B %d inp:", e);
 for (int i=0; i<idim; i++) printf(" %7.5f", data_in[i+e*idim]);
 printf("\n");
}
#endif

#ifdef BLAS
 if (eff_bsize>1)
 { // BLAS block mode: GEMM
  int e,o;
  float alpha = 1.0f;
  float beta = 1.0f;
  REAL *optr, *bptr;

  GpuCopyVectorToMatrix(data_out, b, eff_bsize, odim);

  cublas_stat = cublasSgemm(cublas_handle, CUBLAS_OP_N, CUBLAS_OP_N, odim, eff_bsize, idim,
&alpha, w, odim, data_in, idim, &beta, data_out, odim);
  if (cublas_stat != CUBLAS_STATUS_SUCCESS)
  {
          fprintf(stderr, "!!!! kernel execution error.\n");
    Error();
  }
 }
 else
 { // BLAS vector mode: GEMV
  call_gemv (data_out, w, data_in, b, odim, idim);
 }
```

Fig. 6.76. Modified MachLin::Forw(int eff_bsize) in MachLin.cpp

```
#else
 for (int e=0; e<eff_bsize; e++) {
    // simple matrix vector multiply, TODO: verify bsize
    // TODO: W is stored in BLAS (Fortan) format: colum major !!
  //cout << "forw ex " << e << endl;
   REAL *wptr=w;
   for (int o=0; o<odim; o++) {
    REAL s=b[o];
    for (int i=0; i<idim; i++) s+=wptr[i*odim+o]*data_in[i+e*idim];
    data_out[o+e*odim]=s;
   }
 }
#endif
 nb_forw += eff_bsize;

#if 0
for (int e=0; e<eff_bsize; e++) {
 printf("B %d out:", e);
 for (int i=0; i<odim; i++) printf(" %7.5f", data_out[i+e*odim]);
 printf("\n\n");
}
#endif

}
```

Fig. 6.76. Continued

NVIDIA offers a BLAS library equivalent for use on their GPUs called CUDA BLAS (cuBLAS) [33]. For the GPU implementation the *call_gemm* matrix multiplication operation using the BLAS/OpenBLAS (CPU) libraries in Figure 6.75 were replaced with the *cublasSgemm* function shown in Figure 6.76. The forward pass matrix multiplication described by Thompson et al. [1] is no longer performed by the host. Rather, the GPU in the Tegra K1 is now performing the forward pass matrix multiplication.

Figure 6.77 shows the original *MachLin::Backw(const float lrate, const float wdecay, int eff_bsize)* function definition in MachLin.cpp. The update to the bias vector and the two computationally intensive matrix multiplication operations for the backward pass described by Thompson et al. [1] are performed in this function using the BLAS/OpenBLAS *gemm* function, as shown in red in Figure 6.77 .

```
void MachLin::Backw(const float lrate, const float wdecay, int eff_bsize)
{
 static REAL real1=1.0, real0=0.0;
 static char transN='N', transT='T';
 REAL epsilon = 1.0 + lrate * wdecay;

 if (eff_bsize<=0) eff_bsize=bsize;
 if (!grad_out)
  Error("MachLin::Backw(): output gradient is not set");

#if 0
for (int e=0; e<eff_bsize; e++) {
 printf(" B %d grad:", e);
 for (int i=0; i<idim; i++) printf(" %7.5f", grad_out[i]);
 printf("\n");
}
#endif

  // update bias vector:   b = b + lrate * grad_out
  // NO weight decay
 REAL *gptr = grad_out;
 for (int e=0; e<eff_bsize; e++) {
  REAL *aptr = b;
  for (int i=0; i<odim; i++) *aptr++ += lrate * *gptr++;
 }

#if 0
 printf("b after update:\n");
 for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
 printf("\n");
#endif

  // backprop gradient:   grad_in   =       w'       *   grad_out
  //                 idim x bsize = (odim x idim)'  *  odim x bsize
//printf("GEMM(%lx=%lx * % x)\n",grad_in, w, grad_out);
 GEMM (&transT, &transN, &idim, &eff_bsize, &odim,
     &real1, w, &odim, grad_out, &odim,
     &real0, grad_in, &idim);

  // update weights including weight decay
  // w = lrate  *grad_out * data_in^T + epsilon * w
  // gemm (transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc )
  //                              Go     Din          W
  //       C = alpha*A * B + beta * b
  //
#if 0
 printf("W before update:\n");
 for (int od=0;od<odim;od++) {
```

Fig. 6.77. Original MachLin::Backw(const float lrate, const float wdecay, int eff_bsize) in MachLin.cpp

```
   for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
   printf("\n");
  }
#endif
//printf("GEMM(%lx=%lx * % x)\n",w, grad_out, data_in);
  GEMM (&transN, &transT, &odim, &idim, &eff_bsize,
       &lrate, grad_out, &odim, data_in, &idim,
       &epsilon, w, &odim);
#if 0
  printf("W after update:\n");
  for (int od=0;od<odim;od++) {
   for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
   printf("\n");
  }
#endif
  nb_backw += eff_bsize;
}
```

Fig. 6.77. Continued

Figure 6.78 shows the modifications to the *MachLin::Backw(const float lrate, const float wdecay, int eff_bsize)* function definition im MachLin.cpp for the GPU implementation. The static *char transN* and *transT* were deleted since they were no longer required for the GPU implementation. The bias vector update shown in Figure 6.77 was replaced with the *GpuBatchedAXPY* function, which replaces the nested *for* loops that were used to update the bias vector *b* described by Thompson et al. [1] with the equivalent operation performed by the GPU. The *GpuBatchedAXPY* function prototype statement appears in the Gpu.cuh header file (Appendix D.6) that was added for the GPU implementation, and the function definition and associated kernel call are located in the Gpu.cu file (Appendix D.3) described in Section 6.5.2.

```
void MachLin::Backw(const float lrate, const float wdecay, int eff_bsize)
{
 static REAL real1=1.0, real0=0.0;

 REAL epsilon = 1.0 + lrate * wdecay;

 if (eff_bsize<=0) eff_bsize=bsize;
 if (!grad_out)
  Error("MachLin::Backw(): output gradient is not set");

#if 0
for (int e=0; e<eff_bsize; e++) {
 printf(" B %d grad:", e);
 for (int i=0; i<idim; i++) printf(" %7.5f", grad_out[i]);
 printf("\n");
}
#endif
 REAL *gptr = grad_out;
 GpuBatchedAXPY(odim,lrate,grad_out,1,b,1,eff_bsize);

#if 0
 printf("b after update:\n");
 for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
 printf("\n");
#endif
 int n2A = idim * odim;
 int n2B = odim * eff_bsize;
 /* Performs operation using cublas */
 cublas_stat = cublasSgemm(cublas_handle, CUBLAS_OP_T, CUBLAS_OP_N, idim, eff_bsize, odim, &real1,
w, odim, grad_out, odim, &real0,
 grad_in, idim);
 if (cublas_stat != CUBLAS_STATUS_SUCCESS)
 {
            fprintf(stderr, "!!!! kernel execution error.\n");
            Error();
 }

#if 0
 printf("W before update:\n");
 for (int od=0;od<odim;od++) {
  for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
  printf("\n");
 }
#endif
cublas_stat = cublasSgemm(cublas_handle, CUBLAS_OP_N, CUBLAS_OP_T, odim, idim, eff_bsize, &lrate,
grad_out, odim, data_in, idim, &epsilon, w, odim);
if (cublas_stat != CUBLAS_STATUS_SUCCESS)
```

Fig. 6.78. Modified MachLin::Backw(const float lrate, const float wdecay, int eff_bsize) in MachLin.cpp

```
 {
            fprintf(stderr, "!!!! kernel execution error.\n");
            Error();
 }
#if 0
 printf("W after update:\n");
 for (int od=0;od<odim;od++) {
  for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
  printf("\n");
 }
#endif
 nb_backw += eff_bsize;
}
```

Fig. 6.78. Continued

The computationally intensive backward pass matrix multiplications shown red as GEMM function calls in Figure 6.77 used to update the gradient and weight matrices descriped by Thompson et al [1] were replaced with the cuBLAS equivalent operations using *cublasSgemm*, as shown in red in Figure 6.78, for the GPU implementation [33].

6.5.10 MachPar.cpp

Figure 6.79 shows the original include preprocessor directives for the MachPar.cpp file. Figure 6.80 shows the modification to MachPar.cpp include preprocessor directives for the GPU implementation.

```
using namespace std;
#include <iostream>

#include "Tools.h"
#include "MachTab.h"
#include "MachPar.h"
```

Fig. 6.79. Original include preprocessor directivesfor MachPar.cpp

The modifications to the header files allow the MachPar.cpp file access to the NPP and CUDA Runtime APIs through the addition of the npps.h and cuda_runtim.h header files respectively [33] [34].

```
#include "Tools.h"
#include "MachTab.h"
#include "MachPar.h"
#include "npps.h"
#include "cuda_runtime.h"
```

Fig. 6.80. Modified header includes for MachPar.cpp

Figure 6.81 shows the original *MachPar::do_alloc()* function definition that allocated memory for the *data_out* and *grad_in* variables.

```
void MachPar::do_alloc()
{
 if (data_out) delete [] data_out;
 if (grad_in) delete [] grad_in;
 data_out = (odim*bsize>0) ? new REAL[odim*bsize] : NULL;
 grad_in = (idim*bsize>0) ? new REAL[idim*bsize] : NULL;
}
```

Fig. 6.81. Original MachPar::do_alloc() in MachPar.cpp

The modification to the *MachPar::do_alloc()* function definition for the GPU implementation is shown in Figure 6.82, in which the memory allocations for *data_out* and *grad_in* were accomplished using the *cudaMallocManaged* memory allocation function to enable the use of unified memory.

```
void MachPar::do_alloc()
{
        cudaMallocManaged(&data_out, odim*bsize*sizeof(REAL));
        cudaMallocManaged(&grad_in, idim*odim*sizeof(REAL));
}
```

Fig. 6.82. Modified MachPar::do_alloc() in MachPar.cpp

Figure 6.83 shows the original *MachPar::Forw(int eff_bsize)* function definition in MachPar.cpp where the *memcpy* function was used to copy data from *data_in* to *data_out*.

```
// forward pass for all machines and copy output into cumulated output
void MachPar::Forw(int eff_bsize)
{
 if (machs.empty())
   Error("called Forw() for an empty parallel machine");

 if (eff_bsize<=0) eff_bsize=bsize;

    // we need to set the pointers to the input data of indiv machines
    // one after each other since this depends on the effective bsize !

 REAL *iptr=data_in;
 REAL *optr=data_out;
 for (unsigned int m=0; m<machs.size(); m++) {
  machs[m]->SetDataIn(iptr);
  machs[m]->Forw(eff_bsize);
  memcpy(optr, machs[m]->GetDataOut(), eff_bsize*machs[m]->GetOdim()*sizeof(REAL));
  iptr += eff_bsize*machs[m]->GetIdim();
  optr += eff_bsize*machs[m]->GetOdim();
 }
 nb_forw += eff_bsize;
}
```

Fig. 6.83. Original MachPar::Forw(int eff_bsize) in MachPar.cpp

Figure 6.84 shows the modification to *MachPar::Forw(int eff_bsize)* replacing the memcpy host function with the NPP *nppsCopy_32f* function to perform the copy of data from *data_in* to *data_out* on the GPU.

```
// forward pass for all machines and copy output into cumulated output
void MachPar::Forw(int eff_bsize)
{
 if (machs.empty())
   Error("called Forw() for an empty parallel machine");

 if (eff_bsize<=0) eff_bsize=bsize;

    // we need to set the pointers to the input data of indiv machines
    // one after each other since this depends on the effective bsize !
 REAL *iptr=data_in;
 REAL *optr=data_out;
 for (unsigned int m=0; m<machs.size(); m++)
 {
  machs[m]->SetDataIn(iptr);
  machs[m]->Forw(eff_bsize);
  nppsCopy_32f(machs[m]->GetDataOut(),optr,eff_bsize*machs[m]->GetOdim());
  iptr += eff_bsize*machs[m]->GetIdim();
  optr += eff_bsize*machs[m]->GetOdim();
 }
 nb_forw += eff_bsize;
}
```

Fig. 6.84. Modified MachPar::Forw(int eff_bsize) in MachPar.cpp

Figure 6.85 shows the original *MachPar::Backw(const float lrate, const float wdecay, int eff_bsize)* function definition in MachPar.cpp where the data variable *grad_in* is updated with the values of data variable *grad_out* through the use of the *memcpy* operation on the host.

```
// backward pass for all machines and copy input gradient into cumulated gradient
void MachPar::Backw(const float lrate, const float wdecay, int eff_bsize)
{
 if (machs.empty())
   Error("called Backw() for an empty parallel machine");
 if (eff_bsize<=0) eff_bsize=bsize;

    // we need to set the pointers to output gradients of indiv machines
    // one after each other since this depends on the effective bsize !

 REAL *gptr=grad_in;
 REAL *optr=grad_out;
 for (unsigned int m=0; m<machs.size(); m++) {
  machs[m]->SetGradOut(optr);
  machs[m]->Backw(lrate,wdecay,eff_bsize);
  memcpy(gptr, machs[m]->GetGradIn(), eff_bsize*machs[m]->GetIdim()*sizeof(REAL));
  optr += eff_bsize*machs[m]->GetOdim();
  gptr += eff_bsize*machs[m]->GetIdim();
 }
 nb_backw += eff_bsize;
}
```

Fig. 6.85. Original MachPar::Backw(const float lrate, const float wdecay, int eff_bsize) in MachPar.cpp

An additional change was made to *MachPar::Backw(const float lrate, const float wdecay, int eff_bsize)* in MachPar.cpp for the GPU implementation. With the use of the *cudaMallocManaged* memory allocation of the variables in the GPU implementation that makes use of the unified memory model, memory synchronization is required when the host needs access to the data output from the device. The *cudaDeviceSynchronization* function from the CUDA Runtime API allows the device to safely access to the output data from the GPU [11]. Figure 6.86 shows the addition of the *cudaDeviceSynchronization* function call to *MachPar::Backw(const float lrate, const float wdecay, int eff_bsize)* in MachPar.cpp that synchronizes the managed memory data between the host and the device.

```
// backward pass for all machines and copy input gradient into cumulated gradient
void MachPar::Backw(const float lrate, const float wdecay, int eff_bsize)
{
 if (machs.empty())
   Error("called Backw() for an empty parallel machine");
 if (eff_bsize<=0) eff_bsize=bsize;

    // we need to set the pointers to output gradients of indiv machines
    // one after each other since this depends on the effective bsize !
 REAL *gptr=grad_in;
 REAL *optr=grad_out;
 for (unsigned int m=0; m<machs.size(); m++) {
  machs[m]->SetGradOut(optr);
  machs[m]->Backw(lrate,wdecay,eff_bsize);
  nppsCopy_32f(machs[m]->GetGradIn(),gptr,eff_bsize*machs[m]->GetIdim());
  optr += eff_bsize*machs[m]->GetOdim();
  gptr += eff_bsize*machs[m]->GetIdim();
 }
 cudaDeviceSynchronize();
 nb_backw += eff_bsize;
}
```

Fig. 6.86. Modified MachPar::Backw(const float lrate, const float wdecay, int eff_bsize) in MachPar.cpp

6.5.11 MachSoftmax.cpp

Figure 6.87 shows the include preprocessor directives for Schwenk's original MachSoftmax.cpp file. Figure 6.88 shows the modifications to the include preprocessor directives of MachSoftmax.cpp for the GPU implementation.

```
using namespace std;
#include <iostream>
#include <math.h>

#include "Tools.h"
#include "MachSoftmax.h"
#include "Blas.h"
```

Fig. 6.87. Original include include preprocessor directives in MachSoftmax.cpp

These modifications included the addition of the my_cuda.h and Gpu.cuh (Appendix D.6) preprocessor directives, which are discussed in Sections 6.5.3 and 6.5.1, respectively. Section 6.5.6.

```
using namespace std;
#include <iostream>
#include "math.h"
#include "my_cuda.h"
#include "Tools.h"
#include "MachSoftmax.h"
#include "Blas.h"
#include "Gpu.cuh"
```

Fig. 6.88. Modified include include preprocessor directivesfor MachSoftmax.cpp

Figure 6.89 shows the original *MachSoftmax::Forw(int eff_bsize)* function definition in MachSoftmax.cpp that calculated the softmax value, as described by Thompson et al. [1]. Figure 6.90 shows the modification to *the MachSoftmax::Forw(int eff_bsize)* function definition for the GPU implementation.

```
void MachSoftmax::Forw(int eff_bsize)
{
 if (eff_bsize<=0) eff_bsize=bsize;
 MachLin::Forw(eff_bsize);

  // apply exp() on output and normalize
#ifdef BLAS_INTEL_MKL
 int s=eff_bsize*odim;
 VEXP(&s,data_out,data_out);
 REAL *optr=data_out;
 for (int b=0; b<eff_bsize; b++) {
  REAL sum=0; // TODO: double
  for (int i=0; i<odim; i++) sum += *optr++;
  optr-=odim;
  sum = 1.0/sum;  // circumvent division in loop
  for (int i=0; i<odim; i++) *optr++ *= sum;
 }
#else
 REAL *optr=data_out;
 for (int b=0; b<eff_bsize; b++) {
  REAL sum=0; // TODO: double
  for (int i=0; i<odim; i++) {
   *optr = exp(*optr);
   sum += *optr++;
  }
  optr-=odim;
  sum = 1.0/sum;  // circumvent division in loop
  for (int i=0; i<odim; i++) *optr++ *= sum;
 }
#endif
}
```

Fig. 6.89. Original MachSoftmax::Forw(int eff_bsize) in MachSoftmax.cpp

As shown in Figure 6.90, the two loops used to perform the softmax function in Figure 6.89 have been replaced with the *GpuMachSoftmaxForw* function call. The *GpuMachSoftmaxForw* function performs the softmax operation described by Thompson et al. on the GPU [1]. The function prototype statement for *GpuMachSoftmaxForw* is listed in Gpu.cuh (Appendix D.6); the function definition in is Gpu.cu (Appendix D.3).

```
void MachSoftmax::Forw(int eff_bsize)
{
 if (eff_bsize<=0) eff_bsize=bsize;
 MachLin::Forw(eff_bsize);

 // apply exp() on output and normalize
#ifdef BLAS_INTEL_MKL
 int s=eff_bsize*odim;
 VEXP(&s,data_out,data_out);
 REAL *optr=data_out;
 for (int b=0; b<eff_bsize; b++) {
  REAL sum=0; // TODO: double
  for (int i=0; i<odim; i++) sum += *optr++;
  optr-=odim;
  sum = 1.0/sum;  // circumvent division in loop
  for (int i=0; i<odim; i++) *optr++ *= sum;
 }
#else
 GpuMachSoftmaxForw(bsize,odim,data_out);
#endif
}
```

Fig. 6.90. Modified MachSoftmax::Forw(int eff_bsize) in MachSoftmax.cpp

6.5.12 MachTab.cpp

Figure 6.91 shows the original include preprocessor directives for Schwenk's MachTab.cpp CPU host file. Figure 6.92 shows the modified include preprocessor directives for the GPU implementation.

```
using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
#include "MachTab.h"
```

Fig. 6.91. Original include include preprocessor directivesfor MachTab.cpp

The GPU implementation requires the cuda_runtime.h, Gpu.cuh, and npps.h headers added to MachTab.cpp as shown in Figure 6.92. Section 6.5.4, Section 6.5.6, and Section 6.5.10 describe the use of these header files.

```
using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
#include "MachTab.h"
#include "cuda_runtime.h"
#include "Gpu.cuh"
#include "npps.h"
```

Fig. 6.92. Modified include preprocessor directivesfor MachTab.cpp

Figure 6.93 shows the original *MachTab::do_alloc()* function definition in MachTab.cpp that allocates the memory for the variable *t*. Figure 6.94 shows the modified version of the *MachTab::do_alloc()* memory allocation function for the GPU implementation that performed the memory allocation for the variable *t* with the *cudaMallocManaged* CUDA Runtime API.

```
void MachTab::do_alloc()
{
 if (!ext_alloc) {
   t = new REAL[idim*odim];
   if (!t) Error ("can't allocate memory for table look-up machine");
 }
 else
   ;
}
```

Fig. 6.93. Original MachTab::do_alloc() in MachTab.cpp

```
void MachTab::do_alloc()
{
 if (!ext_alloc) {
   cudaMallocManaged(&t, idim*odim*sizeof(REAL));
   if (!t) Error ("can't allocate memory for table look-up machine");
 }
 else
   ;
}
```

Fig. 6.94. Modified MachTab::do_alloc() in MachTab.cpp

Figure 6.95 shows the original class destructor *MachTab::~MachTab()* that deallocats the memory allocated for variable *t*.

```
MachTab::~MachTab()
{
 if (!ext_alloc & (t!=NULL)) delete [] t;
}
```

Fig. 6.95. Original MachTab::~MachTab() in MachTab.cpp

Figure 6.96 shows the modification to the *MachTab::~MachTab()* deallocation function for the GPU implementation using the *cudaFree* function from the CUDA Runtime API.

```
MachTab::~MachTab()
{
 cudaFree(t);
}
```

Fig. 6.96. Modified MachTab::~MachTab() in MachTab.cpp

Figure 6.97 shows the original *MachTab::Forw(int eff_bsize)* function definition in MachTab.cpp that performs the table update in the forward pass as described by Thompson et al. [1]. Figure 6.98 shows the modification to the *MachTab::Forw(int eff_bsize)* function definition for the GPU implementation.

```
void MachTab::Forw(int eff_bsize)
{
 if (!data_in)
   Error("MachTab::Forw(): input data is not set");

 if (eff_bsize<=0) eff_bsize=bsize;

 REAL *optr=data_out;
 for (int b=0; b<eff_bsize; b++) {
  int idx= (int) data_in[b];
  if (idx<0 || idx>=idim) {
   fprintf(stderr,"ERROR: illegal index (%d) in table look-up machine, should be in [0,%d[\n", idx, idim);
   Error();
  }
  memcpy(optr, t+idx*odim, odim*sizeof(REAL));
  optr+=odim;
 }
 nb_forw+=eff_bsize;
}
```

Fig. 6.97. Original MachTab::Forw(int eff_bsize) in MachTab.cpp

The modified *MachTab::Forw(int eff_bsize)* function definition in Figure 6.98 shows the addition of the *cudaDeviceSynchronize* CUDA Runtime API described in Section 6.5.10. The for loop shown in Figure 6.98 that updates the *idx* variable with *data_in* before the *memcpy* operation is replaced by the *GpuMachTabForw* function call to perform the equivalent operation on the GPU. The function prototype statement for *GpuMachTabForw* occurs in Gpu.cuh (Appendix D.6); the function definition is in Gpu.cu (Appendix D.3).

```
void MachTab::Forw(int eff_bsize)
{
 if (!data_in)
   Error("MachTab::Forw(): input data is not set");

 if (eff_bsize<=0) eff_bsize=bsize;
 cudaDeviceSynchronize();
 GpuMachTabForw(eff_bsize, odim, data_in, t, data_out);

 nb_forw+=eff_bsize;

}
```

Fig. 6.98. Modified MachTab::Forw(int eff_bsize) in MachTab.cpp

Figure 6.99 shows the original *MachTab::Backw(const float lrate, const float wdecay, int eff_bsize)* function definition for updating the table in the backward pass as described by Thompson et al. [1].

```
void MachTab::Backw(const float lrate, const float wdecay, int eff_bsize)
{
 // table[wid] = table[wid] + lrate * grad_out[wid] * data_in[wid]

 REAL *gptr = grad_out;
 for (int b=0; b<eff_bsize; b++) {
  int idx= (int) data_in[b];
  if (idx<0 || idx>=idim) {
   fprintf(stderr,"ERROR: illegal index (%d) in table look-up machine (backw), should be in [0,%d[\n", idx,
idim);
   Error();
  }
  REAL *tptr=t+idx*odim;
#ifdef DEBUG
  printf("  B %d idx=%d\n",b,idx);
  printf("  grad:"); for (int i=idx-2;i<=idx+2;i++) printf(" %f",gptr[i]); printf("\n");
  printf("   tab:");  for (int i=-2;i<=2;i++) printf(" %f",tptr[i]); printf("\n");
#endif
  for (int i=0; i<odim; i++) *tptr++ += lrate * *gptr++;
  grad_in[b]=0; // we don't backprop to the input of a tbale look-up machine
 }
}
```

Fig. 6.99. Original MachTab::Backw(const float lrate, const float wdecay, int eff_bsize) in MachTab.cpp

Figure 6.100 shows the modified version of the *MachTab::Backw(const float lrate, const float wdecay, int eff_bsize)* for the GPU implementation that has the inclusion of the *cudaDeviceSynchronize* function descibed in Section 6.5.10.

```
void MachTab::Backw(const float lrate, const float wdecay, int eff_bsize)
{
 // table[wid] = table[wid] + lrate * grad_out[wid] * data_in[wid]
 cudaDeviceSynchronize();
 REAL *gptr = grad_out;
 for (int b=0; b<eff_bsize; b++) {
  int idx= (int) data_in[b];
  if (idx<0 || idx>=idim) {
   fprintf(stderr,"ERROR: illegal index (%d) in table look-up machine (backw), should be in [0,%d[\n", idx,
idim);
   Error();
  }
  REAL *tptr=t+idx*odim;
#ifdef DEBUG
  printf("  B %d idx=%d\n",b,idx);
  printf("  grad:"); for (int i=idx-2;i<=idx+2;i++) printf(" %f",gptr[i]); printf("\n");
  printf("   tab:");  for (int i=-2;i<=2;i++) printf(" %f",tptr[i]); printf("\n");
#endif
  for (int i=0; i<odim; i++) *tptr++ += lrate * *gptr++;
  grad_in[b]=0; // we don't backprop to the input of a tbale look-up machine
 }
}
```

Fig. 6.100. Modified MachTab::Backw(const float lrate, const float wdecay, int eff_bsize) in MachTab.cpp

6.5.13 MachTanh.cpp

Figure 6.101 shows the original include preprocessor directives for Schwenk's MachTanh.cpp file. Figure 6.102 shows the modification to to the MachTanh.cpp include preprocessor directives to include my_cuda.h and the cuda_runtime.h headers described in Section 6.5.3 and Section 6.5.4 respectively.

```
using namespace std;
#include <iostream>
#include <math.h>

#include "Tools.h"
#include "MachTanh.h"
#include "Blas.h"
```

Fig. 6.101. Original include preprocessor directives in MachTanh.cpp

Figure 6.103 shows the original *MachTanh(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int p_nbbw)* class definition within MachTanh.cpp.

```
using namespace std;
#include <iostream>
#include <math.h>

#include "my_cuda.h"
#include "MachTanh.h"
#include "Blas.h"
#include "cuda_runtime.h"
```

Fig. 6.102. Modified include preprocessor directives in MachTanh.cpp

Figure 6.104 shows the modified *MachTanh(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int p_nbbw)* constructor initialization within MachTanh.cpp for the GPU implementation. The *cudaMalloc* is a CUDA Runtime API for memory allocation of the *tmp_tanh* array on the device [11].

```
MachTanh::MachTanh(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int
p_nbbw)
 : MachLin(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
}
```

Fig. 6.103. Original MachTanh::MachTanh(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int p_nbbw) in MachTanh.cpp

```
MachTanh::MachTanh(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int
p_nbbw)
 : MachLin(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
        cudaMalloc((void **)&tmp_tanh, odim*bsize * sizeof(REAL));
}
```

Fig. 6.104. Modified MachTanh::MachTanh(const int p_idim, const int p_odim, const int p_bsize, const int p_nbfw, const int p_nbbw) in MachTanh.cpp

Figure 6.105 shows the original *Mach::~MachTanh()* class deconstructor within MachTanh.cpp. Figure 6.106 shows the modified *MachTanh::~MachTanh()* class

deconstructor where the *cudaFree* memory deallocation CUDA Runtime API is called to deallocate the *tmp_tanh* variable that was added for the GPU implementation.

```
MachTanh::~MachTanh()
{
}
```

Fig. 6.105. Original MachTanh::~MachTanh() in MachTanh.cpp

```
MachTanh::~MachTanh()
{
        cudaFree(tmp_tanh);
}
```

Fig. 6.106. Modified MachTanh::~MachTanh() in MachTanh.cpp

Figure 6.107 shows the original *MachTanh::Forw(int eff_bsize)* function definition within MachTanh.cpp that performs the tanh operation described by Thompson et al. in the forward pass [1]. Figure 6.108 shows the modification to the *MachTanh::Forw(int eff_bsize)* function definition for the GPU implementation where the for loop where the tanh operation was being performed has been replaced by multiple NPP API function call to perform the tanh operation on the GPU.

```
void MachTanh::Forw(int eff_bsize)
{
 if (eff_bsize<=0) eff_bsize=bsize;
 MachLin::Forw(eff_bsize);

  // apply tanh() on output
#ifdef BLAS_INTEL_MKL
 int s=eff_bsize*odim;
 VTANH(&s,data_out,data_out);
#else
 for (int i=0; i<eff_bsize*odim; i++) data_out[i]=tanh(data_out[i]);
#endif
}
```

Fig. 6.107. Original MachTanh::Forw(int eff_bsize) in MachTanh.cpp

```
void MachTanh::Forw(int eff_bsize)
{

  if (eff_bsize<=0) eff_bsize=bsize;
  MachLin::Forw(eff_bsize);

   // apply tanh() on output
#ifdef BLAS_INTEL_MKL
 int s=eff_bsize*odim;
 VTANH(&s,data_out,data_out);
#else
 int s=eff_bsize*odim;
 nppsMulC_32f_I(2.0,data_out,s);              // 2*x
 nppsExp_32f_I(data_out,s);                          // exp(2*x)
 nppsAddC_32f(data_out,1.0,tmp_tanh,s); // tmp=exp(2*x)+1
 nppsSubC_32f_I(1.0,data_out,s);              // exp(2*x)-1
 nppsDiv_32f_I(tmp_tanh,data_out,s);                // (exp(2*x)-1) / (exp(2*x)+1)

#endif
}
```

Fig. 6.108. Modified MachTanh::Forw(int eff_bsize) in MachTanh.cpp

The original *MachTanh::Backw(const float lrate, const float wdecay, int eff_bsize)* function definition within MachTanh.cpp for the backward pass is shown in Figure 6.109. The *for* loop shown in Figure 6.109 performs the backward pass update for the tanh activation function described by Thompson et al. [1].

```
void MachTanh::Backw(const float lrate, const float wdecay, int eff_bsize)
{
   // derivate tanh activation function
   // multiply grad_hidden by derivatives of hidden layer activities (tanh)
   // grad_out = grad_out .* f'(data_out)
   //          = grad_out .* ( 1 - data_out^2 )

 REAL *aptr = data_out;
 REAL *gptr = grad_out;

 if (eff_bsize<=0) eff_bsize=bsize;
 if (!grad_out)
   Error("MachTanh::Backw(): output gradient is not set");
 for (int i=0; i<odim*eff_bsize; i++) {
   REAL val = *aptr++;
   *gptr++ *= (1.0 - val * val);
 }

 MachLin::Backw(lrate, wdecay, eff_bsize);
}
```

Fig. 6.109. Original MachTanh::Backw(const float lrate, const float wdecay, int eff_bsize) in MachTanh.cpp

Figure 6.110 shows the modified *MachTanh::Backw(const float lrate, const float wdecay, int eff_bsize)* function definition within MachTanh.cpp for the GPU implementation where the *for* loop in Figure 6.109 has been replaced by the NPP API functions to perform the equivalent opereation on the GPU.

```
void MachTanh::Backw(const float lrate, const float wdecay, int eff_bsize)
{

 REAL *aptr = data_out;
 REAL *gptr = grad_out;

 if (eff_bsize<=0) eff_bsize=bsize;
           int d=odim*eff_bsize;
           nppsSqr_32f_I(data_out,d);
           nppsSubCRev_32f_I(1.0,data_out,d);
         nppsMul_32f_I(data_out,grad_out,d);

           MachLin::Backw(lrate, wdecay, eff_bsize);
}
```

Fig. 6.110. Modified MachTanh::Backw(const float lrate, const float wdecay, int eff_bsize) in MachTanh.cpp

6.5.14 MachTanh.h

Figure 6.111 shows the Schwenk's original MachTanh class definition in MachTanh.h. Figure 6.112 shows the modification to the MachTanh class definition where the *tmp_tanh* pointer declaration is made for the GPU implementation.

```
class MachTanh : public MachLin
{
public:
 MachTanh(const int=0, const int=0, const int=1, const int=0, const int=0);
 virtual ~MachTanh();
 virtual int GetMType() {return file_header_mtype_tanh;};     // get type of machine
 virtual void Info(bool=false, char *txt=(char*)"");     // display (detailed) information on machine
 virtual void Forw(int=0);  // calculate outputs for current inputs
   // backprop gradients from output to input and update all weights
 virtual void Backw (const float lrate, const float wdecay, int=0);
};
```

Fig. 6.111. Original MachTanh.h class declaration in MachTanh.h

```
class MachTanh : public MachLin
{
protected:
          REAL *tmp_tanh;
public:
 MachTanh(const int=0, const int=0, const int=1, const int=0, const int=0);
 virtual ~MachTanh();
 virtual int GetMType() {return file_header_mtype_tanh;};     // get type of machine
 virtual void Info(bool=false, char *txt=(char*)"");     // display (detailed) information on machine
 virtual void Forw(int=0);  // calculate outputs for current inputs
   // backprop gradients from output to input and update all weights
 virtual void Backw (const float lrate, const float wdecay, int=0);
};
```

Fig. 6.112. Modified MachTanh.h class declaration in MachTanh.h

6.5.15 Trainer.cpp

The original include preprocessor directivesfor Schwenk's Trainer.cpp is shown in Figure 6.113. The modified include preprocessor directivesfor the GPU implementation is shown in Figure 6.114. The addition of the cuda_runtime.h header allows the use of the CUDA Runtime API as described in Section 6.5.4.

```
using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "Mach.h"
#include "Trainer.h"
```

Fig. 6.113. Original include preprocessor directives in Trainer.cpp

```
using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "Mach.h"
#include "Trainer.h"
#include "cuda_runtime.h"
```

Fig. 6.114. Modified include preprocessor directives inTrainer.cpp

The original *Trainer::Trainer(Mach *pmach, ErrFct *perrfct, char *train_fname,char *dev_fname,REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,int p_maxep, int p_ep)* constructor initialization within Trainer.cpp where the memory allocation for the data variables *buf_input* and *buf_target* is performed is shown in Figure 6.115.

```
Trainer::Trainer (Mach *pmach, ErrFct *perrfct, char *train_fname,char *dev_fname,REAL p_lr_beg, REAL
p_lr_mult, REAL p_wd,int p_maxep, int p_ep)
 : mach(pmach), errfct(perrfct),
   lrate_beg(p_lr_beg), lrate_mult(p_lr_mult), wdecay(p_wd),
   nb_epoch(p_ep), max_epoch(p_maxep)
{
  char     msg[1024];

  idim=mach->GetIdim(); odim=mach->GetOdim(); bsize=mach->GetBsize();
  if (train_fname) {
    data_train = new Data(train_fname);

    if (idim != data_train->GetIdim()) {
      sprintf(msg,"Trainer: input dimension of the training data (%d) does not match the one of the machine
(%d)\n", data_train->GetIdim(), idim);
      Error(msg);
    }
    if (odim != data_train->GetOdim()) {
      sprintf(msg,"Trainer: ouput dimension of the training data (%d) does not match the one of the
machine (%d)\n", data_train->GetOdim(), odim);
      Error(msg);
    }
  }
  else
    data_train=NULL;

  if (dev_fname) {
    data_dev = new Data(dev_fname);
    if (idim != data_dev->GetIdim())
      Error("Trainer: input dimension of the validation data does not match the one of the machine\n");
    if (odim != data_dev->GetOdim())
      Error("Trainer: output dimension of the validation data does not match the one of the machine\n");
    }
  else
    data_dev=NULL;

  buf_input = new REAL[idim*bsize];
  buf_target = new REAL[odim*bsize];
  // memory for the output gradient is allocated by the error function
}
```

Fig. 6.115. Original Trainer::Trainer(Mach *pmach, ErrFct *perrfct, char *train_fname,char *dev_fname,REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,int p_maxep, int p_ep) in Trainer.cpp

Figure 6.116 shows the modification to the *Trainer::Trainer(Mach *pmach, ErrFct *perrfct, char *train_fname,char *dev_fname,REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,int p_maxep, int p_ep)* constructor initialization in Trainer.cpp for the GPU

implementaiton where the memory allocation functions of data variables *buf_input* and *buf target* have been replaced with the *cudaMallocManaged* CUDA Runtime API functions.

```
Trainer::Trainer (Mach *pmach, ErrFct *perrfct,
                          char *train_fname, char *dev_fname,
                          REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,
                          int p_maxep, int p_ep)
 : mach(pmach), errfct(perrfct),
  lrate_beg(p_lr_beg), lrate_mult(p_lr_mult), wdecay(p_wd),
  nb_epoch(p_ep), max_epoch(p_maxep)
{
 char     msg[1024];

 idim=mach->GetIdim(); odim=mach->GetOdim(); bsize=mach->GetBsize();
 if (train_fname) {
  data_train = new Data(train_fname);

  if (idim != data_train->GetIdim()) {
   sprintf(msg,"Trainer: input dimension of the training data (%d) does not match the one of the machine
(%d)\n", data_train->GetIdim(), idim);
   Error(msg);
  }
  if (odim != data_train->GetOdim()) {
   sprintf(msg,"Trainer: ouput dimension of the training data (%d) does not match the one of the
machine (%d)\n", data_train->GetOdim(), odim);
   Error(msg);
  }
 }
 else
  data_train=NULL;

 if (dev_fname) {
  data_dev = new Data(dev_fname);
  if (idim != data_dev->GetIdim())
   Error("Trainer: input dimension of the validation data does not match the one of the machine\n");
  if (odim != data_dev->GetOdim())
   Error("Trainer: output dimension of the validation data does not match the one of the machine\n");
  }
 else
  data_dev=NULL;


 cudaMallocManaged(&buf_input, idim*bsize*sizeof(REAL));

 cudaMallocManaged(&buf_target, odim*bsize*sizeof(REAL));
 // memory for the output gradient is allocated by the error function
}
```

Fig. 6.116. Modified Trainer::Trainer(Mach *pmach, ErrFct *perrfct, char *train_fname,char *dev_fname,REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,int p_maxep, int p_ep) in Trainer.cpp

Figure 6.117 shows the original class deconstructor ~Trainer::Trainer() within Trainer.cpp where the deallocation of the memory for the data variables *buf_input* and *buf_target* is performed. Figure 6.118 shows the modification to the *Trainer::~Trainer()* class deconstructor function for the GPU implementation, where the memory deallocation for the *buf_input* and *buf_target* data variables is now performed using the *cudaFree* memory deallocation.

```
Trainer::~Trainer ()
{
 if (data_train) delete data_train;
 if (data_dev) delete data_dev;
 delete [] buf_input;
 delete [] buf_target;
}
```

Fig. 6.117. Original Trainer::~Trainer() in Trainer.cpp

```
Trainer::~Trainer ()
{
 cudaFree(buf_input);
 cudaFree(buf_target);
}
```

Fig. 6.118. Modified Trainer::~Trainer() in Trainer.cpp

6.5.16 TrainerNgram.cpp

Figure 6.119 shows the original include preprocessor directives in Schwenk's TrainerNgram.cpp. Figure 6.120 shows the modified version of the TrainerNgram.cpp include preprocessor directives with the addition of the my_cuda.h and cuda_runtime.h header files for the GPU implemenation.

```
using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "Mach.h"
#include "TrainerNgram.h"
```

Fig. 6.119. Original include preprocessor directives in TrainerNgram.cpp

Figure 6.120 shows the addition of the *cublasHandle_t cublas_handle* pointer type definition and *cublasStatus_t cublas_stat* type definition for the CUBLAS library API use [33].

```
using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>
#include "my_cuda.h"
#include "Tools.h"
#include "Mach.h"
#include "TrainerNgram.h"
#include "cuda_runtime.h"

cublasHandle_t cublas_handle;
cublasStatus_t cublas_stat;
```

Fig. 6.120. Modified include preprocessor directives in TrainerNgram.cpp

The original *TrainerNgram::Train()* function definition within TrainerNgram.cpp is shown in Figure 6.121. The modified version of the *TrainerNgram::Train()* function definiton for the GPU implementation is shown in Figure 6.122. Figure 6.122 shows the addition of the *cudaDeviceSynchronize* CUDA Runtime API funciton call at the completion of the training process described by Thompson et al. [1] so that the host CPU can safely access the data output from the GPU [11]. Figure 6.122 also shows the use of the *memcpy* function to update the managed memory variables *buf_input* and *buf_target*. By using managed memory for the *buf_input* and *buf_target* variables, the host CPU can update these variables for the GPU device to use thus avoiding explicit memory transfers that could add to execution time [11].

```
REAL TrainerNgram::Train()
{
 if (!data_train) return -1;
#ifdef DEBUG
 printf("*****************\n");
 printf("TrainerNgram::Train():\n");
 printf(" -  data_in: %p \n", (void*) buf_input);
 printf(" -   target: %p \n", (void*) buf_target);
 printf(" - grad_out: %p \n", (void*) errfct->GetGrad());
#endif
 data_train->Rewind();

 REAL log_sum=0;
 nb_ex=0;
 mach->SetDataIn(buf_input);
 mach->SetGradOut(errfct->GetGrad());
 errfct->SetOutput(mach->GetDataOut());
 errfct->SetTarget(buf_target);
 bool data_available;
 do {
    // get a bunch of data
    // TODO: exlude out of slist
  int n=0;
  data_available = true;
  while (n < mach->GetBsize() && data_available) {
    data_available = data_train->Next();
    if (!data_available) break;
    memcpy(buf_input  + n*idim, data_train->input, idim*sizeof(REAL));
    memcpy(buf_target + n*1, data_train->target, 1*sizeof(REAL));
    n++;
  }

  //if (nb_ex%1024==0) printf("."); fflush (stdout);

  if (n>0) {
    mach->Forw(n);
    log_sum += errfct->CalcGrad(n);
#ifdef DEBUG2
    int t=(int) data_train->target[0];
printf("OUTPUT:") ; for (int i=t-2;i<=t+2; i++) printf(" %f",mach->GetDataOut()[i]); printf("\n");
printf("TARGET:") ; for (int i=0;i<1; i++) printf(" %f",data_train->target[i]); printf("\n");
printf("  GRAD:") ; for (int i=t-2;i<=t+2; i++) printf(" %f",errfct->GetGrad()[i]); printf("\n");
#endif
    SetLrate();
    mach->Backw(lrate, wdecay, n);
  }

  nb_ex += n;
 } while (data_available);
```

Fig. 6.121. Original TrainerNgram::Train() in TrainerNgram.cpp

```
 if (nb_ex>0) return exp(-log_sum / (REAL) nb_ex);  // return perplexity

 return -1;
}
```

Fig. 6.121. Continued

```
REAL TrainerNgram::Train()
{
 if (!data_train) return -1;
#ifdef DEBUG
 printf("*****************\n");
 printf("TrainerNgram::Train():\n");
 printf(" -  data_in: %p \n", (void*) buf_input);
 printf(" -   target: %p \n", (void*) buf_target);
 printf(" - grad_out: %p \n", (void*) errfct->GetGrad());
#endif
 data_train->Rewind();

 REAL log_sum=0;
 nb_ex=0;
 mach->SetDataIn(buf_input);
 mach->SetGradOut(errfct->GetGrad());
 errfct->SetOutput(mach->GetDataOut());
 errfct->SetTarget(buf_target);
 bool data_available;
 do {
    // get a bunch of data
    // TODO: exlude out of slist
  int n=0;
  data_available = true;
  while (n < mach->GetBsize() && data_available) {
    data_available = data_train->Next();
    if (!data_available) break;
    memcpy(buf_input  + n*idim, data_train->input, idim*sizeof(REAL));
    memcpy(buf_target + n*1, data_train->target, 1*sizeof(REAL));
    n++;
  }

  //if (nb_ex%1024==0) printf("."); fflush (stdout);

  if (n>0) {
    mach->Forw(n);
    log_sum += errfct->CalcGrad(n);
#ifdef DEBUG2
    int t=(int) data_train->target[0];
printf("OUTPUT:") ; for (int i=t-2;i<=t+2; i++) printf(" %f",mach->GetDataOut()[i]); printf("\n");
printf("TARGET:") ; for (int i=0;i<1; i++) printf(" %f",data_train->target[i]); printf("\n");
printf("  GRAD:") ; for (int i=t-2;i<=t+2; i++) printf(" %f",errfct->GetGrad()[i]); printf("\n");
#endif
    SetLrate();
    mach->Backw(lrate, wdecay, n);
  }

  nb_ex += n;
 } while (data_available);
```

Fig. 6.122. Modified TrainerNgram::Train() in TrainerNgram.cpp

```
cudaDeviceSynchronize();

if (nb_ex>0) return exp(-log_sum / (REAL) nb_ex);  // return perplexity

return -1;
}
```

Fig. 6.122. Continued

The original *TrainerNgram::TestDev(char *fname)* function definition is shown in Figure 6.123. Figure 6.124 shows the modification to the *TrainerNgram::TestDev(char *fname)* function definition for the GPU Implementation. Figure 6.124 shows the addition of two *cudaDeviceSynchronize* CUDA Runtime API funciton calls in the validation process described by Thompson et al. [1] so that the host CPU can safely access the data ouput by the GPU device in the Tegra K1 [11]. Figure 6.124 also shows the use of the *memcpy* function to update the managed memory variables *buf_input* and *buf_target*. The use of managed memory for the *buf_input* and *buf_target* variables avoided additional explicit memory transfers that could have added to execution time [11].

```
REAL TrainerNgram::TestDev(char *fname)
{
 if (!data_dev) return -1;

 if (fname) {
  Error("not yet implemented");
 }

 int nb_ex_dev=0;
 REAL log_sum=0;
 data_dev->Rewind();
 mach->SetDataIn(buf_input);
 errfct->SetOutput(mach->GetDataOut());
 errfct->SetTarget(buf_target);
 bool data_available;
 do {
    // get a bunch of data
    // TODO: exlude out of slist
  int n=0;
  data_available = true;
  while (n < mach->GetBsize() && data_available) {
    data_available = data_dev->Next();
    if (!data_available) break;
    memcpy(buf_input  + n*idim, data_dev->input, idim*sizeof(REAL));
    memcpy(buf_target + n*1, data_dev->target, 1*sizeof(REAL));
    n++;
  }

    // process the bunch
  if (n>0) {
#ifdef DEBUG
printf("in:"); for (int i=0;i<idim;i++) printf(" %f", buf_input[i]);
printf("-> trg:"); for (int i=0;i<1;i++) printf(" %f", buf_target[i]); printf("\n");
#endif
    mach->Forw(n);
    log_sum += errfct->CalcValue(n);
    if (fname) {
      Error();          // TODO: we should get access to the parts of bsize
    }
  }

  nb_ex_dev += n;
 } while (data_available);

 if (nb_ex_dev>0) return exp(-log_sum / (REAL) nb_ex_dev);  // return perplexity
 return -1;
}
```

Fig. 6.123. Original TrainerNgram::TestDev(char *fname) in TrainerNgram.cpp

```
REAL TrainerNgram::TestDev(char *fname)
{
 if (!data_dev) return -1;

 if (fname) {
  Error("not yet implemented");
 }

 int nb_ex_dev=0;
 REAL log_sum=0;
 data_dev->Rewind();
 mach->SetDataIn(buf_input);
 errfct->SetOutput(mach->GetDataOut());
 errfct->SetTarget(buf_target);
 bool data_available;
 do {
   // get a bunch of data
   // TODO: exlude out of slist
  int n=0;
  data_available = true;
  while (n < mach->GetBsize() && data_available) {
   data_available = data_dev->Next();
   if (!data_available) break;
   memcpy(buf_input  + n*idim, data_dev->input, idim*sizeof(REAL));
   memcpy(buf_target + n*1, data_dev->target, 1*sizeof(REAL));
   n++;
  }

   // process the bunch
  if (n>0) {
#ifdef DEBUG
printf("in:"); for (int i=0;i<idim;i++) printf(" %f", buf_input[i]);
printf("-> trg:"); for (int i=0;i<1;i++) printf(" %f", buf_target[i]); printf("\n");
#endif
   cudaDeviceSynchronize();
   mach->Forw(n);
   log_sum += errfct->CalcValue(n);
   if (fname) {
    Error();        // TODO: we should get access to the parts of bsize
   }
  }

  nb_ex_dev += n;
 } while (data_available);
 cudaDeviceSynchronize();
 if (nb_ex_dev>0) return exp(-log_sum / (REAL) nb_ex_dev);  // return perplexity
 return -1;
}
```

Fig. 6.124. Modified TrainerNgram::TestDev(char *fname) in TrainerNgram.cpp

Figure 6.125 shows the original *TrainerNgram::TrainAndTest*() function definition and Figure 6.126 shows the modifications made to the *TrainerNgram::TrainAndTest()* function definiton for the GPU implementation. Figure 6.126 shows the *cublasCreate* function call used to initialize the CUBLAS library prior to any CUBLAS API function call being made [33]. Figure 6.126 also shows the addition of the *cublasDestroy* function call at the completion of the *TrainAndTest* function. The *cublasDestroy* function call releases the hardware resources used by the CUBLAS libraries [33].

```
void TrainerNgram::TrainAndTest ()
{
 if (!data_train) {
   cout << "No training data specified, training impossible" << endl;
   return;
 }

 const int hlen=256;
 char hostname[hlen];
 gethostname(hostname, hlen); hostname[hlen-1]=0;
 cout << "Starting training on host " << hostname << " pid " << getpid() << endl;
 cout << " - training on " << data_train->GetFname() << endl;
 if (data_dev)
   cout << " - validation on " << data_dev->GetFname() << endl;
 cout << " - stopping training at " << max_epoch << " epochs" << endl;
 mach->Info();

 while (!Converged()) {
   InfoPre();
   err_train = Train();
   InfoPost();

   cout << " - starting validation ..."; cout.flush();
   err_dev = TestDev();
   if (err_dev<0)
     cout << " avrg error: no examples !?" << endl;
   else
     cout << " avrg error: " << err_dev << endl;
 }
 cout << "Training stopped" << endl;
 mach->Info();
 //mach->Write();
}
```

Fig. 6.125. Original TrainerNgram::TrainAndTest() in TrainerNgram.cpp

```
void TrainerNgram::TrainAndTest ()
{
 if (!data_train) {
  cout << "No training data specified, training impossible" << endl;
  return;
 }

 cublas_stat = cublasCreate(&cublas_handle);
 if (cublas_stat != CUBLAS_STATUS_SUCCESS)
 {
            fprintf(stderr,"\n CuBLAS Initialization Failed \n");
            cudaDeviceReset();
            Error();
 };

 const int hlen=256;
 char hostname[hlen];
 gethostname(hostname, hlen); hostname[hlen-1]=0;
 cout << "Starting training on host " << hostname << " pid " << getpid() << endl;
 cout << " - training on " << data_train->GetFname() << endl;
 if (data_dev)
  cout << " - validation on " << data_dev->GetFname() << endl;
 cout << " - stopping training at " << max_epoch << " epochs" << endl;
 mach->Info();

 while (!Converged()) {
  InfoPre();
  err_train = Train();
  InfoPost();

  cout << " - starting validation ..."; cout.flush();
  err_dev = TestDev();
  if (err_dev<0)
   cout << " avrg error: no examples !?" << endl;
  else
   cout << " avrg error: " << err_dev << endl;
 }

 cout << "Training stopped" << endl;
 mach->Info();
 /* Shutdown */

 cublasDestroy(cublas_handle);
 //mach->Write();
}
```

Fig. 6.126. Modified TrainerNgram::TrainAndTest() in TrainerNram.cpp

6.6 Build from Nsight

With the required modifications made to the source files to implement the GPU version of the CSLM algorithm on the Tegra K1, an executable had to be generated from Nsight. From within the cslm_train_cuda_rev2 project in Nsight, “Build Project” is selected as shown in Figure 6.127.

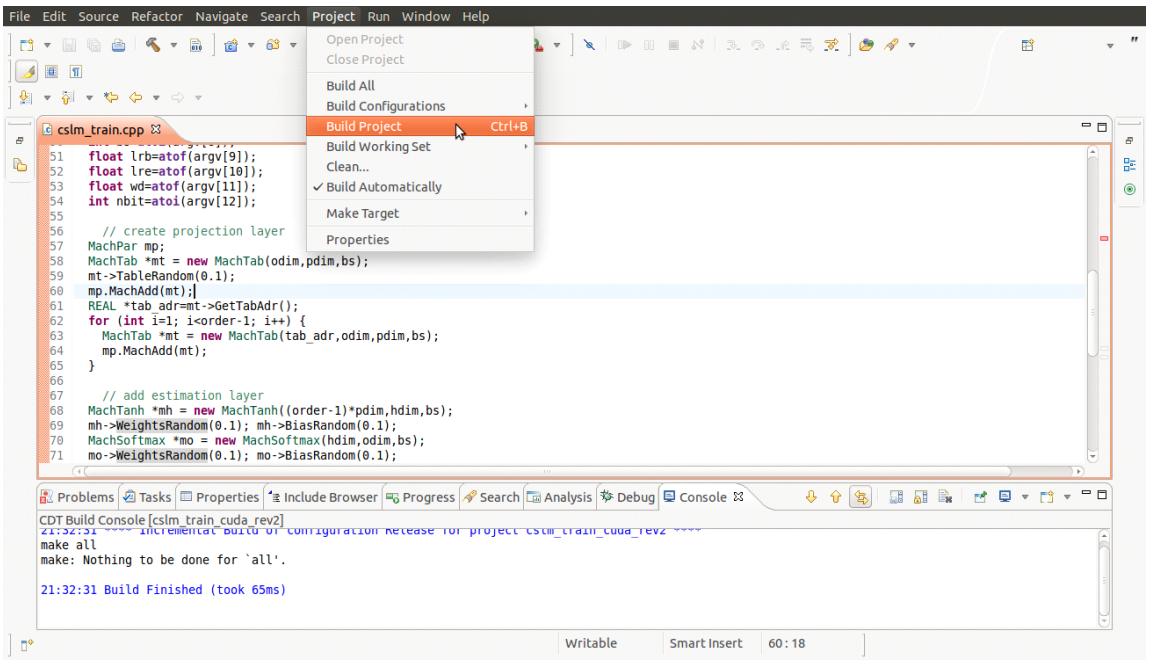

Fig. 6.127. CSLM Train GPU project build.

When the build of the cslm_train completes, the “Console” tab from within Nsight states “Finished Building Target : cslm_train_cuda_rev2” as shown Figure 6.128. The gpu version of the cslm_train was ready to be run on the Tegra K1.

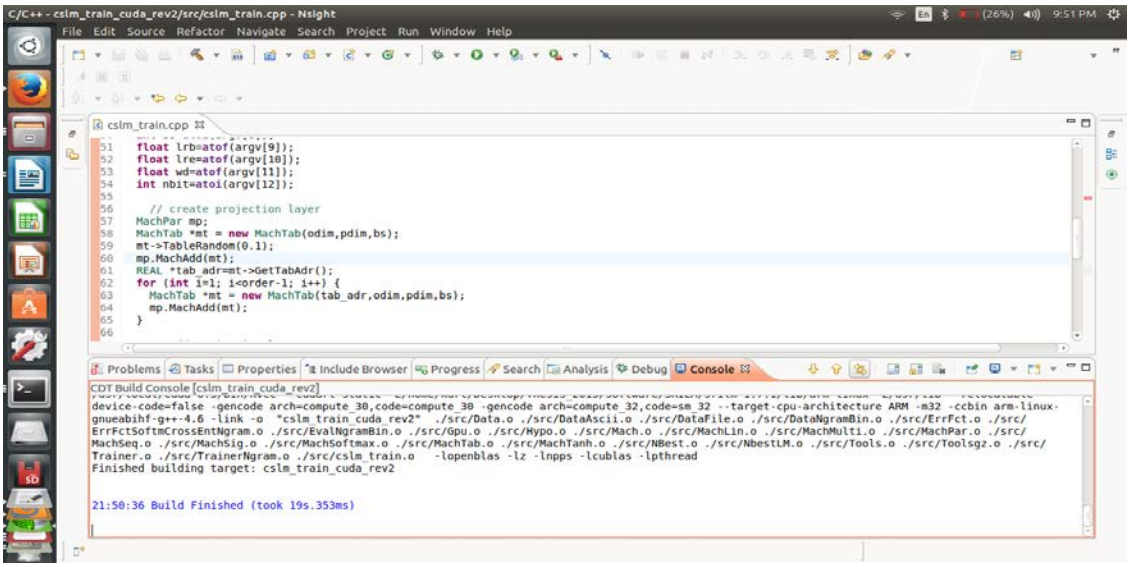

Fig. 6.128. CSLM Train GPU Build Complete

6.7 Execution on Tegra K1

The cslm_train_cuda_rev2 executable had to be copied over to the Jetson TK1 before it could be executed. With Jetson TK1 connected to the router, the executable could be copied over and run remotely. To do this from the laptop a terminal session was launched by using the shortcut “Ctrl+Alt+T” from the keyboard. From the terminal the directory was changed to the directory containing the cslm_train_cuda_rev2 executable using the command shown in Figure 6.129

```
kurt@kurt-VPCCA290X:~$ cd cuda-workspace/cslm_train_cuda_rev2/Release/
```

Fig. 6.129. Directory change to gpu cslm_train build

A gpu_cslm_test directory was remotely created on the Jetson TK1 using the Make Directory (mkdir) command shown in Figure 6.130.

```
kurt@kurt-VPCCA290X:~/cuda-workspace/cslm_train_cuda/Release$ssh ubuntu@10.0.0.2 mkdir
/home/ubuntu/Desktop/gpu_cslm_test;
```

Fig. 6.130. Command to remotely create directory on desktop of Jetson TK1

The cslm_train_cuda_rev2 executable was then transferred over to the Jetson TK1 using the Secure Copy (scp) command, as shown in Figure 6.131.

```
kurt@kurt-VPCCA290X:~/cuda-workspace/cslm_train_cuda/Release$ scp cslm_train_cuda_rev2
ubuntu@10.0.0.2:/home/ubuntu/Desktop/gpu_cslm_test
```

Fig. 6.131. Copy command to copy over the executable to Jetson TK1

When the transfer of the cslm_train_cuda_rev2 executable to the Tegra K1 was complete, the terminal displayed the transfer progress as shown in Figure 6.132.

```
cslm_train_cuda_rev2                          100% 3928KB   3.8MB/s   00:01
kurt@kurt-VPCCA290X:~/cuda-workspace/cslm_train_cuda/Release$
```

Fig. 6.132. Successful transfer of the gpu version of cslm_train to Jetson TK1

Next the files that the cslm_train executable requires had to be transferred over to the Jetson TK1 as well. From the terminal navigate to the cslm example directory on the laptop using the Change Directory (cd) command, as shown in Figure 6.133.

```
kurt@kurt-VPCCA290X:~/cuda-workspace/cslm_train_cuda/Release$ cd
../../../Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram/
```

Fig. 6.133. Directory change to location of required files for cslm on Jetson

From the examples/ngram directory of Schwenk's open source code, the training data file, train.df, and the validation data file, dev.df, were transferred over to the Jetson TK1 using the Secure Copy (scp) command from the terminal prompt as shown in Figure 6.134.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$ scp *.df
ubuntu@10.0.0.2:/home/ubuntu/Desktop/gpu_cslm_test
```

Fig. 6.134. Command for train.df and dev.df transfer from the Sony VPCCA290X laptop to Jetson TK1

Figure 6.135 shows the successful transfer of the dev.df and train.df files to the Jetson TK1. The next file required to be transferred over to the Jetson TK1 was example.mach, also located in the /examples/ngram directory of Schwenk's open source code.

```
dev.df                                        100%  239     0.2KB/s   00:00
train.df                                      100%  568     0.6KB/s   00:00
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$
```

Fig. 6.135. Successful transfer of train.df and dev.df from the Sony VPCCA290X laptop to the Jetson TK1

From the terminal prompt, the command shown in Figure 6.136 was executed to transfer example.mach to the Jetson TK1.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$ scp
example.mach  ubuntu@10.0.0.2:/home/ubuntu/Desktop/gpu_cslm_test
```

Fig. 6.136. Command to copy example.mach to Jetson TK1

Figure 6.137 shows the successful transfer of example.mach to the Jetson TK1. The news08.btxt and news09.bxt binary files also had to be transferred over to the Jetson TK1.

```
example.mach                                  100%   25MB   2.1MB/s   00:12
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$
```

Fig. 6.137. Successful transfer of example.mach to Jetson TK1

Figure 6.138 shows the command from the terminal prompt used to transfer the news08.btxt and news09.btxt files to the Jetson TK1.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$ scp *.btxt
ubuntu@10.0.0.2:/home/ubuntu/Desktop/gpu_cslm_test
```

Fig. 6.138. Command to copy news08.btxt and news09.btxt to Jetson TK1

Figure 6.139 shows the successful transfer of the news08.btxt and news09.btxt to the Jetson TK1. With all of the required files to run the GPU version of the CSLM algorithm

transferred over to the Jetson TK1, it was then necessary to remote into the Jetson TK1 from the laptop in order to remotely execute the cslm_train_cuda_rev2 executable.

```
news08.btxt                                   100%  210KB 210.5KB/s   00:00
news09.btxt                                   100%  276KB 276.0KB/s   00:00
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$
```

Fig. 6.139. Successful transfer of news08.btxt and news09.btxt to Jetson TK1

Figure 6.140 shows the command from the terminal prompt on the laptop used to set up the remote connection to the Jetson TK1.

```
kurt@kurt-VPCCA290X:~/Desktop/cslm_laptop_validation/cslm_v1.0/examples/ngram$ ssh
ubuntu@10.0.0.2
```

Fig. 6.140. Command to remote into the Jetson TK1

Figure 6.141 shows the successful remote connection to the Jetson TK1. Note that the terminal prompt in Figure 6.141 has changed from the laptop's kurt@kurt-VPCCA290X: to the Jetsons ubuntu@tegra-ubuntu:. All terminal commands from this point forward are with respect to the Jetson TK1.

```
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.10.40-ged4f697 armv7l)

 * Documentation:  https://help.ubuntu.com/

Last login: Fri Jan 14 01:37:02 2000 from 10.0.0.3
ubuntu@tegra-ubuntu:~$
```

Fig. 6.141. Successful connection to Jetson TK1

By default the L4T kernel manages the CPU cores for the Tegra K1 using cpuquiet within the device to turn on or off CPU cores depending on the workload. Since the OpenBLAS Tegra K1 version of the CSLM algorithm on the Jetson TK1 used two cores, the GPU version was also allowed the usage of two CPU cores in addition to the GPU core to execute the GPU version of the CLSM algorithm on the Jetson TK1. To do this as before, the command prompt from the terminal was used to navigate to the tegra_cpuquiet directory as shown in Figure 6.142 [32].

```
ubuntu@tegra-ubuntu:cd ../../sys/devices/cpu/cpuquiet/tegra_cpuquiet
```

Fig. 6.142. Changing directory to tegra_cpu quiet

The command at the terminal prompt in Figure 6.143 calls sudo with the /bin/bash command to switch to a root session with the root privleges required to disable cpu_quiet and enable the second CPU core.

```
ubuntu@tegra-ubuntu:/sys/devices/system/cpu/cpuquiet/tegra_cpuquiet$ sudo /bin/bash
[sudo] password for ubuntu:
root@tegra-ubuntu:/sys/devices/system/cpu/cpuquiet/tegra_cpuquiet#
```

Fig. 6.143. Command for root privileges to disable cpu quiet

Figure 6.144 shows the command used at the terminal prompt to disable cpu_quiet.

```
root@tegra-ubuntu:/sys/devices/system/cpu/cpuquiet/tegra_cpuquiet# echo 0 > enable
root@tegra-ubuntu:/sys/devices/system/cpu/cpuquiet/tegra_cpuquiet#
```

Fig. 6.144. Command to disable the cpu_quiet

With cpu_quiet disabled, the command shown in Figure 6.145 was then used at the terminal prompt to change the directory to the cpu1 directory.

```
root@tegra-ubuntu:/sys/devices/system/cpu# cd ../../cpu1
```

Fig. 6.145. Change to cpu1 directory

The command shown in Figure 6.146 was then used at the terminal prompt to manually enable the second CPU core within the Tegra K1.

```
root@tegra-ubuntu:/sys/devices/system/cpu/cpu1# echo 1 > online
root@tegra-ubuntu:/sys/devices/system/cpu/cpu1#
```

Fig. 6.146. Command to enable cpu1

With the second CPU core enabled, the directory was then changed to the location of the cslm_train_cuda_rev2 executable as shown in Figure 6.147.

```
root@tegra-ubuntu:/sys/devices/system/cpu/cpu1# cd
../../../../../home/ubuntu/Desktop/gpu_cslm_test/
```

Fig. 6.147. Change to gpu_cslm_test directory

The LD_LIBRARY_PATH enviroment variable path was updated to include the CUDA 6.5 path for the dynamic libraries used by the cslm_train_cuda_rev2 executable. The command at the terminal prompt shown in Figure 6.148 was used to update the path variable.

```
ubuntu@tegra-ubuntu:~/Desktop/gpu_cslm_test$ export LD_LIBRARY_PATH=/usr/local/cuda-
6.5/lib:$LD_LIBRARY_PATH
```

Fig. 6.148. Setting path variable to cuda path

The cslm_train_cuda_rev2 executable was then lauched using the terminal command shown in Figure 6.149.

```
ubuntu@tegra-ubuntu:~/Desktop/gpu_cslm_test$ ./cslm_train_cuda_rev2 train.df dev.df
example.mach 256 192 14024 4 128 5e-3 4e-7 3e-5 10
```

Fig. 6.149. Execution of the GPU CLSM train on the Jetson TK1

The cslm_train_cuda_rev2 executable runs to the completion of the training and validation process, and the text output from the terminal is saved in a text file as before for comparison later. An example of the output from the cslm_train_cuda_rev2 execution is shown in Figure 6.150. The training and validation portion of the algorithm took about 4 minutes and 40 seconds with an average error of 109.356. Note that the error values recorded at the end of each epoch in Figure 6.150 are nearly identical to those of Figures 5.25, 5.31, and 6.38, which depict results of the BLAS version on the Sony VPCCA290X laptop, the OpenBLAS version on the Sony VPCCA290X laptop, and the OpenBLAS CPU version on the Tegra K1, respectively.

```
ubuntu@tegra-ubuntu:~/Desktop/Thesis/cslm_v1.0/examples/ngram$ ./cslm_train_cuda_rev2 train.df
dev.df example.mach 256 192 14024 4 128  5e-3 4e-7 3e-5 10
Sequential machine [3] 3- .. -14024, bs=128, passes=0/0
  Parallel machine 3- .. 768, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
    MachTab 1[14024]-256, bs=128, passes=0/0
  MachTanh 768-192, bs=128, passes=0/0
  MachSoftmax 192-14024, bs=128, passes=0/0
Opening data description 'train.df'
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Summary of used data:
 - news09.btxt  1.0000 *     63070 =     63070
 - total number of examples: 63070
 - resampling with seed 12345678
 - all resampling coefficients are set to one, loading data once
 - loading all data into memory
 - shuffling data 10 times ... done
Opening data description 'dev.df'
 - news09.btxt binary ngram file with 65595 words in 2525 lines, order=4, mode=3
    counting ... 63070 4-grams (0 unk, 5050 ignored)
Summary of used data:
 - news09.btxt  1.0000 *     63070 =     63070
 - total number of examples: 63070
 - resampling with seed 12345678
 - all resampling coefficients are set to one, loading data once
 - loading all data into memory
Starting training on host tegra-ubuntu pid 1885
 - training on train.df
 - validation on dev.df
 - stopping training at 10 epochs
 - Sequential machine [3] 3- .. -14024, bs=128, passes=0/0
 -   Parallel machine 3- .. 768, bs=128, passes=0/0
 -     MachTab 1[14024]-256, bs=128, passes=0/0
 -     MachTab 1[14024]-256, bs=128, passes=0/0
 -     MachTab 1[14024]-256, bs=128, passes=0/0
 -   MachTanh 768-192, bs=128, passes=0/0
 -   MachSoftmax 192-14024, bs=128, passes=0/0
Starting epoch 1 at Sat Jan  1 01:01:50 2000
 - intial lrate=5.0000e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 950.158
 - starting validation ... avrg error: 566.79
Starting epoch 2 at Sat Jan  1 01:02:18 2000
 - intial lrate=4.7598e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 522.23
 - starting validation ... avrg error: 420.934
Starting epoch 3 at Sat Jan  1 01:02:46 2000
 - intial lrate=4.5417e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
```

Fig. 6.150. CSLM CPU/GPU Results on Tegra K1 platform

```
- epoch finished, 63070 examples seen, average error: 407.591
 - starting validation ... avrg error: 332.251
Starting epoch 4 at Sat Jan  1 01:03:14 2000
 - intial lrate=4.3427e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 335.862
 - starting validation ... avrg error: 284.998
Starting epoch 5 at Sat Jan  1 01:03:42 2000
 - intial lrate=4.1603e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 283.561
 - starting validation ... avrg error: 234.436
Starting epoch 6 at Sat Jan  1 01:04:10 2000
 - intial lrate=3.9927e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 242.749
 - starting validation ... avrg error: 199.289
Starting epoch 7 at Sat Jan  1 01:04:38 2000
 - intial lrate=3.8381e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 208.618
 - starting validation ... avrg error: 170.987
Starting epoch 8 at Sat Jan  1 01:05:06 2000
 - intial lrate=3.6950e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 179.748
 - starting validation ... avrg error: 144.58
Starting epoch 9 at Sat Jan  1 01:05:34 2000
 - intial lrate=3.5621e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 155.224
 - starting validation ... avrg error: 125.404
Starting epoch 10 at Sat Jan  1 01:06:02 2000
 - intial lrate=3.4385e-03, wdecay=3.0000e-05
 - shuffling data 10 times ... done
 - epoch finished, 63070 examples seen, average error: 134.671
 - starting validation ... avrg error: 109.356
Training stopped
 - Sequential machine [3] 3- .. -14024, bs=128, passes=1261400/630700
 -   Parallel machine 3- .. 768, bs=128, passes=1261400/630700
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -     MachTab 1[14024]-256, bs=128, passes=1261400/0
 -   MachTanh 768-192, bs=128, passes=1261400/630700
 -   MachSoftmax 192-14024, bs=128, passes=1261400/630700
```

Fig. 6.150. Continued

6.8 Performance

Table 6.6 summarizes the performance of the CSLM algorithm for various CPU/GPU platforms. It provides the execution time per epoch and average error for the OpenBLAS CPU version executed on the Tegra K1 as described in Section 6.4 and the heterogeneous CPU/GPU version on the Jetson TK1 described in Section 6.7. Also included in Table 6.6 for comparison are the results reported by Thompson et al. [1] using the Quadro FX 5800 desktop GPU mounted on an HP Z800 workstation consisting of 12 Intel Xeon x5600 processors operating at 2.8 GHz . The OpenBLAS Tegra K1 CPU version of the training and validation portion of the CSLM algorithm described in Section 6.4 executed in about 8 minutes and 29 seconds per epoch, and the Tegra K1 GPU implementation discussed in Section 6.7 took about 28 seconds per epoch, which came very close to the performance observed on the Quadro FX 5800 desktop GPU.

Table 6.6
Tegra K1 Implementation Results

Tegra K1 Implementation	Time per epoch	Average Error
Tegra K1 CPU With GPU	28 seconds	109.356
Tegra K1 CPU OpenBLAS	509 seconds	109.356
Quadro FX 5800 [1]	26 seconds	109.368

7. RESULTS

Table 7.1 shows a summary of the results for the various implementations described in this research. The BLAS implementation on the Sony VPCCA290X laptop described in Section 5.6 using a single CPU cores took 12.38 minutes per epoch. The OpenBLAS CPU implementation on the VPCCA290X laptop using 2 CPU cores discussed in Section 5.7 took 2.3 minutes per epoch. The execution time of the MKL implementation on the HP Z800 worksation using 8 of the 12 cores as described by Thompson et al. [1] took 36 seconds per epoch. The OpenBLAS implementation on the Tegra K1 using only two of the four ARM A15 CPUs as described in Section 6.4 took 509 seconds (~8.48 minutes) per epoch. The GPU implementation on the Tegra K1 from Section 6.7 took 28 seconds per epoch.

Table 7.1
Summary of Performance

	Platform	Time per epoch (min)	Average Error
BLAS (single core active)	VPCCA290X laptop 2 core Intel i5-2410M processor 2.3 GHz	12.38	109.359
OpenBLAS (2 cores active)	VPCCA290X laptop 2 core Intel i5-2410M processor 2.3 GHz	2.30	109.356
MKL [1]	HP Z800 workstation 12 Intel Xeon x5660 Processors 2.8 GHz	0.6 [1]	109.359 [1]
Tegra K1 CPU With GPU	Tegra K1 Implementation 2 ARM A15 cores active + GPU	0.47	109.356
Tegra K1 CPU OpenBLAS	Tegra K1 Implementation 2 ARM A15 cores active	8.48	109.356
Quadro FX 5800 Desktop GPU [1]	HP Z800 workstation 12 Intel Xeon x5600 processors 2.8 GHz + GPU	0.43 [1]	109.368 [1]

8. CONCLUSION

The training and validation portion of the Schwenks CSLM algorithm ver 1.0 was implemented on the Tegra K1 heterogenous mobile processor using only the OpenBLAS library and two of the four ARM A15CPU cores available on the device. The performace of this CPU version of the CSLM version 1.0 algorithm on the Tegra K1 was compared to BLAS and OpenBLAS versions of the algorithm on a Sony Intel i5 VPCCA290X laptop. The OpenBLAS version of the CSLM version 1.0 algorithm on the Tegra K1 using two of the four ARM A15 CPU cores did perform better than the single CPU core BLAS version of the CSLM algortim on the Sony VPCCA290X laptop, but the CPU performance of the Tegra K1 was not able to match that of the OpenBLAS version on the Sony VPCCA290X laptop.

A version of the CSLM algorithm using the CPU as well as the GPU core available on the Tegra K1 was then created to see if porting of the computationally intensive portions of the training and validation operations to the GPU could provide any significant reduction in the execution time. The porting to the GPU of the computationally intensive portions of the algorithm provided a significant improvement in performance even though the GPU core in the Tegra K1 consistes of only a single SMX. The execution time of the GPU version of the CSLM training and validation operation was similar to the execution time reported by Thompson et al. [1]. The comparable performance of the GPU on the Tegra K1 compared to that of the desktop GPU used by Thompson et al. [1] for the training and validation operations demonstrates the computational power that the current heterogenous mobile processor possesses with significantly smaller power requirements. The use of unified memory in the Tegra K1 greatly reduced the need for explicit memory copies between the Tegra K1 host CPUs

and the GPU in the Tegra K1, which may have played a role in the Tegra K1 implementation.

The sequential nature of the CSLM algorithm plays a significant role in the performance of the GPU version of the CSLM algorithm on the Tegra K1 compared to destop results reported by Thompson et al. [1]. Given that the Tegra K1 contains only a single SMX, the same increase in performace of another algorithm where a larger degree of parallelism is required would not be possible given the Tegra K1 would not have the GPU resources required.

LIST OF REFERENCES

LIST OF REFERENCES

[1] E. A. Thompson and T. R. Anderson, "A CUDA implementation of the Continuous Space," *J Supercomput*, p. 65–86, 2014.

[2] H. Schwenk, D. D´echelotte and J.-L. Gauvain, "Continuous Space Language Models for Statistical Machine Translation," *Prague Bulletin of Mathematical Linguistics*, no. 93, p. 137–146, January 2010.

[3] H. Schwenk, "Continuous Space Language Models," *Computer Speech and Language*, pp. 491-518, 9 October 2006.

[4] Y. Bengio, R. Ducharme, P. Vincent and C. Jauvin, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research 3*, p. 1137–1155, 2003.

[5] "Introduction to artificial neural networks," in *Electronic Technology Directions to the Year 2000*, 1995.

[6] C. Bishop, Neural Networks for Pattern Recognition, Clarendon Press, Oxford, 1995.

[7] H. Schwenk, "CSLM - A modular Open-Source Continuous Space Language Modeling Toolkit," University of Le Mans, Le Mans, France.

[8] NVIDIA Corporation, "Tegra K1 Whitepaper," 2014. [Online]. Available: http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-K1-whitepaper.pdf. [Accessed 11 3 2016].

[9] NVIDIA, "Jetson TK1 Development Kit Quick Start Guide," 2014. [Online]. Available: http://developer.download.nvidia.com/embedded/jetson/TK1/docs/2_ GetStart/Jeston_TK1_User_Guide.pdf. [Accessed 27 3 2016].

[10] NVIDIA Corporation, "NVIDIA-Kepler-GK110-Architecture-Whitepaper," 2012. [Online]. Available: https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf. [Accessed 27 3 2016].

[11] NVIDIA Corporation, ""CUDA C Programming Guide"," 2015. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/. [Accessed 20 February 2016].

[12] NVIDIA Corporation, ""Tegra K1 Whitepaper"," 2014. [Online]. Available: http://www.nvidia.com/content/pdf/tegra_white_papers/tegra-k1-whitepaper.pdf. [Accessed 7 January 2016].

[13] NVIDIA Corporation, "Kepler_Tuning_Guide," 2015. [Online]. Available: http://docs.nvidia.com/cuda/kepler-tuning-guide/index.html#axzz42YgQ5ndZ. [Accessed 5 3 2016].

[14] Ubuntu, "Download Ubuntu Desktop," 2016. [Online]. Available: http://www.ubuntu.com/download/alternative-downloads. [Accessed 13 3 2016].

[15] H. Schwenk, "CSLML: Continuous Space Language Model Toolkit," 2010. [Online]. Available: http://www-lium.univ-lemans.fr/cslm/. [Accessed 13 3 2016].

[16] "Basic Linear Algebra Subprograms," 2016. [Online]. Available: https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms. [Accessed 3 4 2016].

[17] xianyi, "OpenBLAS," 2014. [Online]. Available: https://sourceforge.net/projects/openblas/files/v0.2.8-arm/. [Accessed 20 3 2016].

[18] "SourceForge," [Online]. Available: http://sourceforge.net/projects/openblas/files/v0.2.8-arm/.

[19] Intel. [Online]. Available: http://ark.intel.com/products/52224/Intel-Core-i5-2410M-Processor-3M-Cache-up-to-2_90-GHz.

[20] "BLAS," 2015. [Online]. Available: http://www.netlib.org/blas/. [Accessed 20 3 2016].

[21] "OpenBLAS wiki faq," [Online]. Available: https://github.com/xianyi/OpenBLAS /wiki/faq#what.

[22] "OpenBLAS Wiki," [Online]. Available: https://github.com/xianyi/ OpenBLAS/wiki.

[23] Ubuntu, "SynapticHowto," 2013. [Online]. Available: https://help.ubuntu.com/community/SynapticHowto. [Accessed 13 3 2016].

[24] SRI. [Online]. Available: http://www.speech.sri.com/projects/srilm/.

[25] NVIDIA Corporation, "Jetson platform brief May2014," 2014. [Online]. Available: http://developer.download.nvidia.com/embedded/jetson/TK1/docs/Jetson_platform_brief_May2014.pdf. [Accessed 13 3 2016].

[26] "elinux.org," 2016. [Online]. Available: http://elinux.org/Jetson_TK1. [Accessed 2 4 2016].

[27] NVIDIA Corporation, "JetPack TK1," 2015. [Online]. Available: http://docs.nvidia.com/jetpack-tk1/1_1/content/developertools/mobile/jetpack_main.htm. [Accessed 1 3 2016].

[28] NVIDIA Corporation, "Download and Install JetPack TK1," 1 3 2015. [Online]. Available: http://docs.nvidia.com/jetpack-tk1/1_1/content/developertools/mobile/jetpack_install.htm. [Accessed 2016].

[29] NVIDIA Corporation, "Remote application development using NVIDIA® Nsight™ Eclipse Edition," 2014. [Online]. Available: https://devblogs.nvidia.com/parallelforall/remote-application-development-nvidia-nsight-eclipse-edition/. [Accessed 3 4 2016].

[30] NVIDIA Corporation, "NVIDIA Nsight Eclipse Edition for Jetson TK1," 2014. [Online]. Available: https://devblogs.nvidia.com/parallelforall/nvidia-nsight-eclipse-edition-for-jetson-tk1/. [Accessed 27 3 2016].

[31] NVIDIA Corporation, "Nsight Eclipse Edition Getting Started Guide," 2015. [Online]. Available: http://docs.nvidia.com/cuda/nsight-eclipse-edition-getting-started-guide/#axzz42GeqfGgi. [Accessed 2 3 2016].

[32] "Jetson Performance," 2015. [Online]. Available: http://elinux.org/Jetson/Performance. [Accessed 20 3 2016].

[33] NVIDIA, "cublas," 2015. [Online]. Available: http://docs.nvidia.com/cuda/cublas/index.html#axzz43pJoJxxS. [Accessed 3 3 2016].

[34] NVIDIA, "NVIDIA NPP Library," 2015. [Online]. Available: http://docs.nvidia.com/cuda/pdf/NPP_Library.pdf. [Accessed 3 3 2016].

[35] NVIDIA Corporation, "CUDA_C_Best_Practices_Guide," 2015. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#axzz44pNAXapZ. [Accessed 1 4 2016].

APPENDICES

A. LAPTOP IMPLEMENTATION SOURCE FILES

A.1 Original CSLM Makefile

```
#
# Example of a simple makefile to build the CSLM librare and tools
#
# rcsid $Id: makefile,v 1.9 2010/01/28 09:26:14 schwenk Exp $
#
OBJS=Tools.o \
        Mach.o MachTab.o \
        MachLin.o MachSig.o MachTanh.o MachSoftmax.o \
        MachMulti.o MachSeq.o MachPar.o \
        Data.o DataFile.o DataAscii.o DataNgramBin.o \
        ErrFct.o ErrFctMSE.o ErrFctMCE.o ErrFctCrossEnt.o ErrFctSoftmCrossEntNgram.o\
        Trainer.o TrainerNgram.o \
        EvalNgramBin.o \
        NBest.o Hypo.o Toolsgz.o NbestLM.o NbestLMSRI.o NbestCSLM.o

TOOLS=cslm_train cslm_eval net_info text2bin nbest

# link with SRILM
# the environment varibale SRILM must be correctly set
LIBS_SRI=-L$(SRILM)/lib/i686 -loolm -ldstruct -lmisc -lz -lm

#
# select which BLAS libray to use
#

# default BLAS available on many LINUX distrubutions
# This is pretty slow and does not support multi-threading
#BLAS=-DBLAS_STD
#LIBS_MKL=/usr/lib64/libblas.so.3

# Intel's MKL libray (http://software.intel.com/en-us/intel-mkl)
# This is usually much faster and supports multi-threading
BLAS=-DBLAS_INTEL_MKL -I/opt/intel/Compiler/11.1/046/mkl/include
LIBS_INTEL=/opt/intel/Compiler/11.1/046
LIBS_MKL=-L$(LIBS_INTEL)/mkl/lib/em64t -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -
L$(LIBS_INTEL)/lib/intel64 -liomp5 -lpthread -lm

LIB_CSLM=libcslm.a
LIBS=$(LIBS_MKL) $(LIBS_SRI)

CC=g++ -mtune=core2
CFLAGS=-Wall -I$(SRILM)/include -g ${DB} ${BLAS
```

```
42
43#CC=icc
44#LD=icc
45#CFLAGS=-Wall -msse4.1 -fast
46
47%.o: %.cpp
48        $(CC) $(CFLAGS) -c $<
49
50all: $(TOOLS)
51
52hperf: hperf.cpp $(LIB_CSLM)
53        ${CC} $(CFLAGS) -o hperf hperf.cpp $(LIB_CSLM) $(LIBS)
54
55$(LIB_CSLM): $(OBJS)
56        ar r $(LIB_CSLM) $(OBJS)
57
58cslm_train: cslm_train.cpp $(LIB_CSLM)
59        ${CC} $(CFLAGS) -o cslm_train cslm_train.cpp $(LIB_CSLM) $(LIBS)
60
61cslm_eval: cslm_eval.cpp $(LIB_CSLM)
62        ${CC} $(CFLAGS) -o cslm_eval cslm_eval.cpp $(LIB_CSLM) $(LIBS)
63
64net_info: net_info.cpp $(LIB_CSLM)
65        ${CC} $(CFLAGS) -o net_info net_info.cpp $(LIB_CSLM) $(LIBS)
67
68text2bin: text2bin.cpp
69        ${CC} $(CFLAGS) -o text2bin text2bin.cpp $(LIBS)
70
71nbest: nbest_cmd.cpp $(LIB_CSLM)
72        ${CC} $(CFLAGS) -o nbest nbest_cmd.cpp $(LIB_CSLM) $(LIBS)
73
74clean:
75        @rm $(OBJS) $(TOOLS) $(LIB_CSLM)
76
77locks: RCS
78        @echo "current RCS locks:\n"
79        @grep 'strict;' RCS/* | grep -v locks
80        @echo ""
81
82diff: RCS
83        @echo "changed files with respect to last RCS branch:\n"
84        @for i in RCS/*,v; \
85          do rcsdiff -q $$i > /dev/null; \
86            if [ $$? = 1 ]; then echo $$i; fi; \
87        done
88        @echo ""
89
90
91# DO NOT DELETE
```

A.2 Modified libblas CSLM Makefile

```
1#
2# Example of a simple makefile to build the CSLM librare and tools
```

```
#
# rcsid $Id: makefile,v 1.9 2010/01/28 09:26:14 schwenk Exp $
#
OBJS=Tools.o \
        Mach.o MachTab.o \
        MachLin.o MachSig.o MachTanh.o MachSoftmax.o \
        MachMulti.o MachSeq.o MachPar.o \
        Data.o DataFile.o DataAscii.o DataNgramBin.o \
        ErrFct.o ErrFctMSE.o ErrFctMCE.o ErrFctCrossEnt.o ErrFctSoftmCrossEntNgram.o\
        Trainer.o TrainerNgram.o \
        EvalNgramBin.o \
        NBest.o Hypo.o Toolsgz.o NbestLM.o NbestLMSRI.o NbestCSLM.o

TOOLS=cslm_train cslm_eval net_info text2bin nbest

# link with SRILM
# the environment varibale SRILM must be correctly set
LIBS_SRI=-L/home/kurt/Desktop/cslm_laptop_validation/srilm-1.6.0/lib/i686-m64 -loolm -
ldstruct -lmisc -lz -lm
SRILM = /home/kurt/Desktop/cslm_laptop_validation/srilm-1.6.0
#
# select which BLAS libray to use
#

# default BLAS available on many LINUX distrubutions
# This is pretty slow and does not support multi-threading
BLAS=-DBLAS_STD
LIBS_MKL=/usr/lib/ libblas.so

# Intel's MKL libray (http://software.intel.com/en-us/intel-mkl)
# This is usually much faster and supports multi-threading
#BLAS=-DBLAS_INTEL_MKL -I/opt/intel/Compiler/11.1/046/mkl/include
#LIBS_INTEL=/opt/intel/Compiler/11.1/046
#LIBS_MKL=-L$(LIBS_INTEL)/mkl/lib/em64t -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -
L$(LIBS_INTEL)/lib/intel64 -liomp5 -lpthread -lm

LIB_CSLM=libcslm.a
LIBS=$(LIBS_MKL) $(LIBS_SRI)

CC=g++ -mtune=core2
CFLAGS=-Wall -I$(SRILM)/include -g ${DB} ${BLAS}

#CC=icc
#LD=icc
#CFLAGS=-Wall -msse4.1 -fast

%.o: %.cpp
        $(CC) $(CFLAGS) -c $<

all: $(TOOLS)

hperf: hperf.cpp $(LIB_CSLM)
        ${CC} $(CFLAGS) -o hperf hperf.cpp $(LIB_CSLM) $(LIBS)

$(LIB_CSLM): $(OBJS)
        ar r $(LIB_CSLM) $(OBJS)
```

```
56
58cslm_train: cslm_train.cpp $(LIB_CSLM)
59        ${CC} $(CFLAGS) -o cslm_train cslm_train.cpp $(LIB_CSLM) $(LIBS)
60
61cslm_eval: cslm_eval.cpp $(LIB_CSLM)
62        ${CC} $(CFLAGS) -o cslm_eval cslm_eval.cpp $(LIB_CSLM) $(LIBS)
63
64net_info: net_info.cpp $(LIB_CSLM)
65        ${CC} $(CFLAGS) -o net_info net_info.cpp $(LIB_CSLM) $(LIBS)
66
67text2bin: text2bin.cpp
68        ${CC} $(CFLAGS) -o text2bin text2bin.cpp $(LIBS)
69
70nbest: nbest_cmd.cpp $(LIB_CSLM)
71        ${CC} $(CFLAGS) -o nbest nbest_cmd.cpp $(LIB_CSLM) $(LIBS)
72
73clean:
74        @rm $(OBJS) $(TOOLS) $(LIB_CSLM)
75
76locks: RCS
77        @echo "current RCS locks:\n"
78        @grep 'strict;' RCS/* | grep -v locks
79        @echo ""
80
81diff: RCS
82        @echo "changed files with respect to last RCS branch:\n"
83        @for i in RCS/*,v; \
84          do rcsdiff -q $$i > /dev/null; \
85            if [ $$? = 1 ]; then echo $$i; fi; \
86        done
87        @echo ""
88
89
90# DO NOT DELETE
```

A.3 Modified OpenBLAS CSLM Makefile

```
1#
2# Example of a simple makefile to build the CSLM librare and tools
3#
4# rcsid $Id: makefile,v 1.9 2010/01/28 09:26:14 schwenk Exp $
5#
6OBJS=Tools.o \
7        Mach.o MachTab.o \
8        MachLin.o MachSig.o MachTanh.o MachSoftmax.o \
9        MachMulti.o MachSeq.o MachPar.o \
10       Data.o DataFile.o DataAscii.o DataNgramBin.o \
11       ErrFct.o ErrFctMSE.o ErrFctMCE.o ErrFctCrossEnt.o ErrFctSoftmCrossEntNgram.o\
12       Trainer.o TrainerNgram.o \
13       EvalNgramBin.o \
14       NBest.o Hypo.o Toolsgz.o NbestLM.o NbestLMSRI.o NbestCSLM.o
15
16TOOLS=cslm_train cslm_eval net_info text2bin nbest
17
```

```
# link with SRILM
# the environment varibale SRILM must be correctly set
LIBS_SRI=-L/home/kurt/Desktop/cslm_laptop_validation/srilm-1.6.0/lib/i686-m64 -loolm -ldstruct -lmisc -lz -lm
SRILM = /home/kurt/Desktop/cslm_laptop_validation/srilm-1.6.0
#
# select which BLAS libray to use
#

# default BLAS available on many LINUX distrubutions
# This is pretty slow and does not support multi-threading
BLAS=-DBLAS_STD
LIBS_MKL=/usr/lib/libopenblas.so

# Intel's MKL libray (http://software.intel.com/en-us/intel-mkl)
# This is usually much faster and supports multi-threading
#BLAS=-DBLAS_INTEL_MKL -I/opt/intel/Compiler/11.1/046/mkl/include
#LIBS_INTEL=/opt/intel/Compiler/11.1/046
#LIBS_MKL=-L$(LIBS_INTEL)/mkl/lib/em64t -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -L$(LIBS_INTEL)/lib/intel64 -liomp5 -lpthread -lm

LIB_CSLM=libcslm.a
LIBS=$(LIBS_MKL) $(LIBS_SRI)

CC=g++ -mtune=core2
CFLAGS=-Wall -I$(SRILM)/include -g ${DB} ${BLAS}

#CC=icc
#LD=icc
#CFLAGS=-Wall -msse4.1 -fast

%.o: %.cpp
	$(CC) $(CFLAGS) -c $<

all: $(TOOLS)

hperf: hperf.cpp $(LIB_CSLM)
	${CC} $(CFLAGS) -o hperf hperf.cpp $(LIB_CSLM) $(LIBS)

$(LIB_CSLM): $(OBJS)
	ar r $(LIB_CSLM) $(OBJS)

cslm_train: cslm_train.cpp $(LIB_CSLM)
	${CC} $(CFLAGS) -o cslm_train cslm_train.cpp $(LIB_CSLM) $(LIBS)

cslm_eval: cslm_eval.cpp $(LIB_CSLM)
	${CC} $(CFLAGS) -o cslm_eval cslm_eval.cpp $(LIB_CSLM) $(LIBS)

net_info: net_info.cpp $(LIB_CSLM)
	${CC} $(CFLAGS) -o net_info net_info.cpp $(LIB_CSLM) $(LIBS)

text2bin: text2bin.cpp
	${CC} $(CFLAGS) -o text2bin text2bin.cpp $(LIBS)

nbest: nbest_cmd.cpp $(LIB_CSLM)
	${CC} $(CFLAGS) -o nbest nbest_cmd.cpp $(LIB_CSLM) $(LIBS)
```

```
72
73clean:
74        @rm $(OBJS) $(TOOLS) $(LIB_CSLM)
75
76locks: RCS
77        @echo "current RCS locks:\n"
78        @grep 'strict;' RCS/* | grep -v locks
79        @echo ""
80
81diff: RCS
82        @echo "changed files with respect to last RCS branch:\n"
83        @for i in RCS/*,v; \
84          do rcsdiff -q $$i > /dev/null; \
85            if [ $$? = 1 ]; then echo $$i; fi; \
86        done
87        @echo ""
88
89
90# DO NOT DELETE
```

A.4 Original SRILM 1.6.0 Makefile

```
#
# Top-level Makefile for SRILM
#
# $Header: /home/srilm/CVS/srilm/Makefile,v 1.64 2011/12/08 19:40:24 stolcke Exp $
#

# SRILM = /home/speech/stolcke/project/srilm/devel
MACHINE_TYPE := $(shell $(SRILM)/sbin/machine-type)

RELEASE := $(shell cat RELEASE)

# Include common SRILM variable definitions.
include $(SRILM)/common/Makefile.common.variables

PACKAGE_DIR = ..

INFO = \
        CHANGES \
        RELEASE \
        README \
        doc \
        Copyright \
        License

MODULES = \
        misc \
        dstruct \
        lm \
        flm \
        lattice \
        utils
```

```
EXCLUDE = \
        me \
        htk \
        contrib \
        lm/src/test \
        flm/src/test \
        lattice/src/test \
        dstruct/src/test \
        utils/src/fsmtest \
        common/COMPILE-HOSTS

VERSION_HEADER = \
        SRILMversion.h

MAKE_VARS = \
        SRILM=$(SRILM) \
        MACHINE_TYPE=$(MACHINE_TYPE) \
        OPTION=$(OPTION) \
        MAKE_PIC=$(MAKE_PIC)

World:  dirs
        $(MAKE) init
        $(MAKE) release-headers
        $(MAKE) depend
        $(MAKE) release-libraries
        $(MAKE) release-programs
        $(MAKE) release-scripts

# build central include directory and scripts only
msvc:   dirs
        $(MAKE) init
        $(MAKE) release-headers
        $(MAKE) release-scripts
        cd utils/src; $(MAKE) $(MAKE_VARS) release


depend-all:     dirs release-headers
        @gawk '!/^#/ { print $$1, $$2, $$3 }' common/COMPILE-HOSTS | sort -u | \
        while read prog host type; do \
                rm -f DONE; (set -x; \
                $$prog $$host "cd $(SRILM); $(MAKE) $(MFLAGS) SRILM=$(SRILM)
MACHINE_TYPE=$$type OPTION=$(OPTION) init depend && touch DONE" < /dev/null); \
                [ -f DONE ] || exit 1; \
        done; rm -f DONE

compile-all:    dirs
        @gawk '!/^#/' common/COMPILE-HOSTS | \
        while read prog host type option; do \
                rm -f DONE; (set -x; \
                $$prog $$host "cd $(SRILM); $(MAKE) $(MFLAGS) SRILM=$(SRILM)
MACHINE_TYPE=$$type OPTION=$$option init release-libraries release-programs && touch
DONE" < /dev/null); \
                [ -f DONE ] || exit 1; \
        done; rm -f DONE

clean-all:      dirs
```

```
	@gawk '!/^#/' common/COMPILE-HOSTS | \
	while read prog host type option; do \
		rm -f DONE; (set -x; \
		$$prog $$host "cd $(SRILM); $(MAKE) $(MFLAGS) SRILM=$(SRILM)
MACHINE_TYPE=$$type OPTION=$$option cleanest && touch DONE" < /dev/null); \
		[ -f DONE ] || exit 1; \
	done; rm -f DONE

dirs:
	-mkdir -p include lib bin

remove-dirs:
	-$(RMDIR) $(SRILM_BINDIR)
	-$(RMDIR) $(SRILM_LIBDIR)
	-$(RMDIR) $(SRILM_BIN)
	-$(RMDIR) $(SRILM_LIB)
	-$(RMDIR) $(SRILM_INCDIR)

init depend all programs release clean cleaner cleanest superclean sanitize desanitize \
release-headers release-libraries release-programs release-scripts:
	for subdir in $(MODULES); do \
		(cd $$subdir/src; $(MAKE) $(MAKE_VARS) $@) || exit 1; \
	done

pristine:
	for subdir in $(MODULES); do \
		(cd $$subdir/src; $(MAKE) $(MAKE_VARS) $@) || exit 1; \
	done
	$(MAKE) $(MAKE_VARS) remove-dirs

test try gzip:
	for subdir in $(MODULES); do \
		[ ! -d $$subdir/test ] || \
		(cd $$subdir/test; $(MAKE) $(MAKE_VARS) $@) || exit 1; \
	done

# files needed for the web page
WWW_DOCS = CHANGES License INSTALL RELEASE
WWW_DIR = /home/spftp/www/DocumentRoot/projects/srilm

www:	$(WWW_DOCS)
	ginstall -m 444 $(WWW_DOCS) $(WWW_DIR)/docs
	ginstall -m 444 man/html/*.[1-9].html $(WWW_DIR)/manpages

TAR = /usr/local/gnu/bin/tar

package:	$(PACKAGE_DIR)/EXCLUDE
	(cd misc/src; $(MAKE) $(MAKE_VARS) $(VERSION_HEADER))
	$(TAR) cvzXf $(PACKAGE_DIR)/EXCLUDE $(PACKAGE_DIR)/srilm-$(RELEASE).tar.gz
.
	rm $(PACKAGE_DIR)/EXCLUDE

package_notest:		$(PACKAGE_DIR)/EXCLUDE
	echo test >> $(PACKAGE_DIR)/EXCLUDE
	$(TAR) cvzXf $(PACKAGE_DIR)/EXCLUDE $(PACKAGE_DIR)/srilm-$(RELEASE)-
notest.tar.gz .
```

```
	rm $(PACKAGE_DIR)/EXCLUDE

package_bin:	$(PACKAGE_DIR)/EXCLUDE-$(MACHINE_TYPE)
	$(TAR) cvhzXf $(PACKAGE_DIR)/EXCLUDE-$(MACHINE_TYPE)
$(PACKAGE_DIR)/srilm-$(RELEASE)-$(MACHINE_TYPE).tar.gz $(INFO) include lib man bin
sbin
	rm -f $(PACKAGE_DIR)/EXCLUDE-$(MACHINE_TYPE)

package_x:
	$(MAKE) $(MAKE_VARS) sanitize
	$(MAKE) $(MAKE_VARS) RELEASE=$(RELEASE)_x package
	$(MAKE) $(MAKE_VARS) desanitize

$(PACKAGE_DIR)/EXCLUDE:	force
	rm -f DONE
	(find bin/* lib/* */bin/* */obj/* */src/test */test/output */test/logs -type d -print -prune ; \
	find $(EXCLUDE) include bin -print; \
	find . \( -name Makefile.site.\* -o -name "*.~[0-9]*" -o -name ".#*" -o -name
Dependencies.\* -o -name core -o -name "core.[0-9]*" -o -name \*.3rdparty -o -name .gdb_history
-o -name out.\* -o -name "*[._]pure[._]*" -o -type l -o -name RCS -o -name CVS -o -name
.cvsignore -o -name GZ.files \) -print) | \
	sed 's,^\./,,' > $@

$(PACKAGE_DIR)/EXCLUDE-$(MACHINE_TYPE):	$(PACKAGE_DIR)/EXCLUDE
	fgrep -l /bin/sh bin/* > $(PACKAGE_DIR)/EXCLUDE-shell
	fgrep -v -f $(PACKAGE_DIR)/EXCLUDE-shell $(PACKAGE_DIR)/EXCLUDE | \
	egrep -v 'include|^bin$$|$(MACHINE_TYPE)[^~]*$$' > $@
	-egrep '$(MACHINE_TYPE).*[._]pure[._]' $(PACKAGE_DIR)/EXCLUDE >> $@
	-egrep '$(MACHINE_TYPE)_[gp]' $(PACKAGE_DIR)/EXCLUDE >> $@
	rm -f $(PACKAGE_DIR)/EXCLUDE-shell

force:
```

A.5 Modified SRILM 1.6.0 Makefile

```
#
# Top-level Makefile for SRILM
#
# $Header: /home/srilm/CVS/srilm/Makefile,v 1.64 2011/12/08 19:40:24 stolcke Exp $
#

SRILM = /home/kurt/Desktop/cslm_laptop_validation/srilm-1.6.0
MACHINE_TYPE := $(shell $(SRILM)/sbin/machine-type)

RELEASE := $(shell cat RELEASE)

# Include common SRILM variable definitions.
include $(SRILM)/common/Makefile.common.variables

PACKAGE_DIR = ..

INFO = \
	CHANGES \
	RELEASE \
```

```
	README \
	doc \
	Copyright \
	License

MODULES = \
	misc \
	dstruct \
	lm \
	flm \
	lattice \
	utils

EXCLUDE = \
	me \
	htk \
	contrib \
	lm/src/test \
	flm/src/test \
	lattice/src/test \
	dstruct/src/test \
	utils/src/fsmtest \
	common/COMPILE-HOSTS

VERSION_HEADER = \
	SRILMversion.h

MAKE_VARS = \
	SRILM=$(SRILM) \
	MACHINE_TYPE=$(MACHINE_TYPE) \
	OPTION=$(OPTION) \
	MAKE_PIC=$(MAKE_PIC)

World:  dirs
	$(MAKE) init
	$(MAKE) release-headers
	$(MAKE) depend
	$(MAKE) release-libraries
	$(MAKE) release-programs
	$(MAKE) release-scripts

# build central include directory and scripts only
msvc:   dirs
	$(MAKE) init
	$(MAKE) release-headers
	$(MAKE) release-scripts
	cd utils/src; $(MAKE) $(MAKE_VARS) release


depend-all:	dirs release-headers
	@gawk '!/^#/ { print $$1, $$2, $$3 }' common/COMPILE-HOSTS | sort -u | \
	while read prog host type; do \
		rm -f DONE; (set -x; \
		$$prog $$host "cd $(SRILM); $(MAKE) $(MFLAGS) SRILM=$(SRILM)
MACHINE_TYPE=$$type OPTION=$(OPTION) init depend && touch DONE" < /dev/null); \
		[ -f DONE ] || exit 1; \
```

```
	done; rm -f DONE

compile-all:	dirs
	@gawk '!/^#/' common/COMPILE-HOSTS | \
	while read prog host type option; do \
		rm -f DONE; (set -x; \
		$$prog $$host "cd $(SRILM); $(MAKE) $(MFLAGS) SRILM=$(SRILM)
MACHINE_TYPE=$$type OPTION=$$option init release-libraries release-programs && touch
DONE" < /dev/null); \
		[ -f DONE ] || exit 1; \
	done; rm -f DONE

clean-all:	dirs
	@gawk '!/^#/' common/COMPILE-HOSTS | \
	while read prog host type option; do \
		rm -f DONE; (set -x; \
		$$prog $$host "cd $(SRILM); $(MAKE) $(MFLAGS) SRILM=$(SRILM)
MACHINE_TYPE=$$type OPTION=$$option cleanest && touch DONE" < /dev/null); \
		[ -f DONE ] || exit 1; \
	done; rm -f DONE

dirs:
	-mkdir -p include lib bin

remove-dirs:
	-$(RMDIR) $(SRILM_BINDIR)
	-$(RMDIR) $(SRILM_LIBDIR)
	-$(RMDIR) $(SRILM_BIN)
	-$(RMDIR) $(SRILM_LIB)
	-$(RMDIR) $(SRILM_INCDIR)

init depend all programs release clean cleaner cleanest superclean sanitize desanitize \
release-headers release-libraries release-programs release-scripts:
	for subdir in $(MODULES); do \
		(cd $$subdir/src; $(MAKE) $(MAKE_VARS) $@) || exit 1; \
	done

pristine:
	for subdir in $(MODULES); do \
		(cd $$subdir/src; $(MAKE) $(MAKE_VARS) $@) || exit 1; \
	done
	$(MAKE) $(MAKE_VARS) remove-dirs

test try gzip:
	for subdir in $(MODULES); do \
		[ ! -d $$subdir/test ] || \
		(cd $$subdir/test; $(MAKE) $(MAKE_VARS) $@) || exit 1; \
	done

# files needed for the web page
WWW_DOCS = CHANGES License INSTALL RELEASE
WWW_DIR = /home/spftp/www/DocumentRoot/projects/srilm

www:	$(WWW_DOCS)
	ginstall -m 444 $(WWW_DOCS) $(WWW_DIR)/docs
	ginstall -m 444 man/html/*.[1-9].html $(WWW_DIR)/manpages
```

```
TAR = /usr/local/gnu/bin/tar

package:	$(PACKAGE_DIR)/EXCLUDE
	(cd misc/src; $(MAKE) $(MAKE_VARS) $(VERSION_HEADER))
	$(TAR) cvzXf $(PACKAGE_DIR)/EXCLUDE $(PACKAGE_DIR)/srilm-$(RELEASE).tar.gz
.
	rm $(PACKAGE_DIR)/EXCLUDE

package_notest:	$(PACKAGE_DIR)/EXCLUDE
	echo test >> $(PACKAGE_DIR)/EXCLUDE
	$(TAR) cvzXf $(PACKAGE_DIR)/EXCLUDE $(PACKAGE_DIR)/srilm-$(RELEASE)-
notest.tar.gz .
	rm $(PACKAGE_DIR)/EXCLUDE

package_bin:	$(PACKAGE_DIR)/EXCLUDE-$(MACHINE_TYPE)
	$(TAR) cvhzXf $(PACKAGE_DIR)/EXCLUDE-$(MACHINE_TYPE)
$(PACKAGE_DIR)/srilm-$(RELEASE)-$(MACHINE_TYPE).tar.gz $(INFO) include lib man bin
sbin
	rm -f $(PACKAGE_DIR)/EXCLUDE-$(MACHINE_TYPE)

package_x:
	$(MAKE) $(MAKE_VARS) sanitize
	$(MAKE) $(MAKE_VARS) RELEASE=$(RELEASE)_x package
	$(MAKE) $(MAKE_VARS) desanitize

$(PACKAGE_DIR)/EXCLUDE:	force
	rm -f DONE
	(find bin/* lib/* */bin/* */obj/* */src/test */test/output */test/logs -type d -print -prune ; \
	find $(EXCLUDE) include bin -print; \
	find . \( -name Makefile.site.\* -o -name "*.~[0-9]*" -o -name ".#*" -o -name
Dependencies.\* -o -name core -o -name "core.[0-9]*" -o -name \*.3rdparty -o -name .gdb_history
-o -name out.\* -o -name "*[._]pure[._]*" -o -type l -o -name RCS -o -name CVS -o -name
.cvsignore -o -name GZ.files \) -print) | \
	sed 's,^\./,,' > $@

$(PACKAGE_DIR)/EXCLUDE-$(MACHINE_TYPE):	$(PACKAGE_DIR)/EXCLUDE
	fgrep -l /bin/sh bin/* > $(PACKAGE_DIR)/EXCLUDE-shell
	fgrep -v -f $(PACKAGE_DIR)/EXCLUDE-shell $(PACKAGE_DIR)/EXCLUDE | \
	egrep -v 'include|^bin$$|$(MACHINE_TYPE)[^~]*$$' > $@
	-egrep '$(MACHINE_TYPE).*[._]pure[._]' $(PACKAGE_DIR)/EXCLUDE >> $@
	-egrep '$(MACHINE_TYPE)_[gp]' $(PACKAGE_DIR)/EXCLUDE >> $@
	rm -f $(PACKAGE_DIR)/EXCLUDE-shell

force:
```

B. SCHWENK'S CSLM TOOLKIT VERSION 1.0 SOURCEFILES

B.1 Blas.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Blas.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Blas_h
#define _Blas_h

#include <string.h>     // memcpy()
#include "Tools.h"


// BLAS helper functions

#ifdef BLAS_INTEL_MKL
  #include "mkl_blas.h"
  #include "mkl_vml.h"
// for single precision
  #define VTANH vstanh_
  #define VEXP vsexp
```

```
  #define VLOG vslog_
  #define GEMV sgemv
  #define GEMM sgemm
#endif

#ifdef BLAS_STD
extern "C" void sgemv_(const char *trans, const int *m, const int *n,
const float *alpha,
                  const float *a, const int *lda, const float *x, const
int *incx,
                  const float *beta, float *y, const int *incy);
extern "C" void sgemm_(const char *transa, const char *transb, const
int *m, const int *n, const int *k,
                  const float *alpha, const float *a, const int *lda,
const float *b, const int *ldb,
                  const float *beta, float *c, const int *ldc);
  #define GEMV sgemv_
  #define GEMM sgemm_
#endif

// matrix/vector multiplication: c = 1.0*A * b + 1.0 * c
// the matrix must be stored in COLUM MAJOR order

/*-------------------------------------------------------------------------
-----*
 *
 * Wrapper routine for GEMV function
 * that uses the TRANSPOSED fortran routine
 *
 * dest = matrix * source + bias
 *
 *   dest: dim_dest x 1
 * matrix: dim_dest x dim_src
 * source: dim_src x 1
 *
 *-------------------------------------------------------------------------
-----*/

inline void call_gemv (REAL *dest, REAL *matrix, REAL *source, REAL
*bias,
                       int dim_dest, int dim_src)
{
      char    trans = 'N';
      REAL    fact = 1.0;
      int         inc = 1;

      // int sgemv(char *trans, integer *m, integer *n,
      //     real *alpha, *real *a, integer *lda,
      //     real *x, integer *incx, real *beta, real *y, *integer
*incy)
      //
      // y := alpha*A*x + beta*y
      //         m x n


      debug("-mkl- call gemv\n");
      memcpy (dest, bias, dim_dest * sizeof(REAL));
```

```
      GEMV (&trans, &dim_dest, &dim_src, &fact, matrix, &dim_dest,
source, &inc, &fact, dest, &inc);
}


// matrix/matrix multiplication: C = alpha*A * B + beta * b
// both must be stored in COLUM MAJOR order

inline void call_gemm (REAL *C, REAL *A, REAL *B, REAL beta, int dimy,
int dimx, int dimk)
{
  char    transN = 'N';
  REAL    alpha = 1.0;

    // gemm ( transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c,
ldc )
    //   * C = alpha*A * B + beta * b
    //              mxn mxk        kxn
    //              lda     ldb  ldc


  TRACE("-mkl- call gemm\n");
  GEMM (&transN, &transN, &dimy, &dimx, &dimk, &alpha, A, &dimy, B,
&dimk, &beta, C, &dimy);
}

#endif
```

B.2 cslm_eval.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: cslm_eval.cpp,v 1.7 2010/01/25 12:53:31 schwenk Exp $
```

```
 */

using namespace std;
#include <iostream>

#include "Mach.h"
#include "DataNgramBin.h"
#include "TrainerNgram.h"
#include "ErrFctSoftmCrossEntNgram.h"


int main (int argc, char *argv[])
{
  int order=4;
  int mode=3;
  if (argc<3 || argc>5) {
    fprintf(stderr,"usage: %s network binary-data order [mode]\n",
argv[0]);
    Error();
  }
  if (argc>3) order=atoi(argv[3]);
  if (order<2) Error("order must be greater than 1");
  if (argc>4) mode=atoi(argv[4]);
  if (mode<1 || mode>15) Error("mode must be in 1..15");

  cout << "Evaluating CSLM: " << argv[1] << endl;
    // read network
  ifstream ifs;
  ifs.open(argv[1],ios::binary);
  CHECK_FILE(ifs,argv[1]);
  Mach *m = Mach::Read(ifs);
  ifs.close();
  m->Info();
#if 0
  m->SetBsize(1);
  REAL idata[]={1,2,3,4};
  m->SetDataIn(idata);
  m->Forw();
  return 0;
#endif

    // creating dummy DataFile
  cout << "Using data:\n";
  DataNgramBin df(argv[2], 1.0, order, mode);
  Data data(df);

    // create a trainer for testing only
  ErrFctSoftmCrossEntNgram errfct(*m);
  TrainerNgram trainer(m,&errfct, data);
  cout << "Perplexity: "; cout.flush();
  cout << trainer.TestDev() << endl;;

  exit(1); // brute force exit since we have trouble with memroy
deallocation

  delete m;
  return 0;
```

```
}
```

B.3 cslm_train.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: cslm_train.cpp,v 1.4 2010/01/25 12:53:31 schwenk Exp $
 *
 * This is a simple program to perform the training of continuous space
LMs
 */

using namespace std;
#include <iostream>

#include "Mach.h"
#include "MachTab.h"
#include "MachTanh.h"
#include "MachSoftmax.h"
#include "MachSeq.h"
#include "MachPar.h"
#include "TrainerNgram.h"
#include "ErrFctSoftmCrossEntNgram.h"

int main (int argc, char *argv[])
{

   // get params
  if (argc != 13) {
    fprintf(stderr,"usage: %s train.df dev.df machine-name dim-proj
dim-hidden dim-output order bsize lrate-begin lrate-mult wdecay nb-
iter\n", argv[0]);
    Error();
  }
```

```
  int pdim=atoi(argv[4]);
  int hdim=atoi(argv[5]);
  int odim=atoi(argv[6]);
  int order=atoi(argv[7]);
  int bs=atoi(argv[8]);
  float lrb=atof(argv[9]);
  float lre=atof(argv[10]);
  float wd=atof(argv[11]);
  int nbit=atoi(argv[12]);

    // create projection layer
  MachPar mp;
  MachTab *mt = new MachTab(odim,pdim,bs);
  mt->TableRandom(0.1);
  mp.MachAdd(mt);
  REAL *tab_adr=mt->GetTabAdr();
  for (int i=1; i<order-1; i++) {
    MachTab *mt = new MachTab(tab_adr,odim,pdim,bs);
    mp.MachAdd(mt);
  }

    // add estimation layer
  MachTanh *mh = new MachTanh((order-1)*pdim,hdim,bs);
  mh->WeightsRandom(0.1); mh->BiasRandom(0.1);
  MachSoftmax *mo = new MachSoftmax(hdim,odim,bs);
  mo->WeightsRandom(0.1); mo->BiasRandom(0.1);
  MachSeq mlp;
  mlp.MachAdd(&mp);
  mlp.MachAdd(mh); mlp.MachAdd(mo);
  mlp.Info();

  ErrFctSoftmCrossEntNgram errfct(mlp);
  TrainerNgram trainer(&mlp, &errfct, argv[1], argv[2], lrb, lre, wd,
nbit);
  //cout << "Initial perplexity: " << trainer.TestDev() << endl;
  trainer.TrainAndTest();

  ofstream fs;
  fs.open(argv[3],ios::binary);
  CHECK_FILE(fs,argv[3]);
  mlp.Write(fs);
  fs.close();

  mp.Delete();
  delete mh;
  delete mo;

  return 0;
}
```

B.4 Data.cpp

```
/*
```

```
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Data.cpp,v 1.12 2010/01/25 12:53:31 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <stdlib.h>

#include "Tools.h"
#include "Data.h"
#include "DataAscii.h"
#include "DataNgramBin.h"

const int   DATA_LINE_LEN=1024;
const char* DATA_HEADER_TXT="DataDescr";
const int   DATA_HEADER_ID=1;
const char* DATA_PRELOAD="Preload";
const int   DATA_PRELOAD_ALWAYS=1;  // order of constants is important
!
const int   DATA_PRELOAD_ONCE=2;
const int   DATA_PRELOAD_DONE=3;
const char* DATA_RESAMPL_MODE="ResamplMode";
const char* DATA_RESAMPL_SEED="ResamplSeed";
const char* DATA_SHUFFLE_MODE="ShuffleMode";
const char* DATA_NORMALIZE_MODE="Normalize";

/*************************
 *
 *************************/

Data::Data(char *p_fname)
 : fname(p_fname), idim(0), odim(0), nb_totl(0),
   preload(0), resampl_mode(0), resampl_seed(1234567890),
shuffle_mode(0),
   norm_mode(0),
```

```
   idx(-1), mem_inp(NULL), mem_trg(NULL), input(NULL), target(NULL)
{

  cout << "Opening data description '" << fname << "'" << endl;
  ifstream ifs;
  ifs.open(fname,ios::in);
  CHECK_FILE(ifs,fname);

    // parsing data description
  int i=ReadInt(ifs,DATA_HEADER_TXT);
  if (i!=DATA_HEADER_ID) Error("unknown data description header\n");

  while (!ifs.eof()) {
    bool ok=false;
    string buf; char line[DATA_LINE_LEN];
    ifs >> buf;
    if (buf[0]=='#') {ifs.getline(line, DATA_LINE_LEN); continue;} //
skip comments
    if (buf=="") break; // HACK
    if (buf==DATA_PRELOAD) { preload=DATA_PRELOAD_ALWAYS; ok=true; }
    if (buf==DATA_RESAMPL_MODE) { ifs >> resampl_mode; ok=true; }
    if (buf==DATA_RESAMPL_SEED) { ifs >> resampl_seed; ok=true; }
    if (buf==DATA_SHUFFLE_MODE) { ifs >> shuffle_mode; ok=true; }
    if (buf==DATA_NORMALIZE_MODE) { ifs >> norm_mode; ok=true; }
    if (buf==DATA_FILE_ASCII) {
      datafile.push_back(new DataAscii(ifs)); ok=true;
    }
    if (buf==DATA_FILE_NGRAMBIN) {
      datafile.push_back(new DataNgramBin(ifs)); ok=true;
    }
    if (datafile.size()==1) {idim=datafile[0]->GetIdim();
odim=datafile[0]->GetOdim(); }
    if (datafile.size()>=1) {
      if (idim != datafile.back()->GetIdim()) Error("mismatch in input
dimension\n");
      if (odim != datafile.back()->GetOdim()) Error("mismatch in output
dimension\n");
    }

    if (!ok) {
      ifs.getline(line, DATA_LINE_LEN);
      cerr << buf << "" << line << endl;
      Error("parse error in above line of the datafile\n");
    }
  }
  ifs.close();

  nb_totl=0;
  cout << "Summary of used data:" << endl;
  for (i=0; i<(int) datafile.size(); i++) nb_totl+=datafile[i]->Info();

  cout << " - total number of examples: " << nb_totl << endl;
  if (resampl_mode) {
    cout << " - resampling with seed " << resampl_seed << endl;
    srand48(resampl_seed);
  }
  if (preload > 0) {
```

```
    mem_inp = new REAL[nb_totl*idim];
    if (odim>0) mem_trg = new REAL[nb_totl*odim];

      // check whether there is a resmpling coeff != 0
      // i.e. we need to resample at each rewind
    float s=0;
    for (vector<DataFile*>::iterator it = datafile.begin();
it!=datafile.end(); ++it)
      s+=(*it)->GetResampl();
    if (s>=datafile.size()) {
      preload=DATA_PRELOAD_ONCE;
      cout << " - all resampling coefficients are set to one, loading
data once\n";
    }

  }
  else {
    if (norm_mode>0)
      Error("Normalization of the data is only implemented with
preloading\n");
  }
  Preload();
  Shuffle();
}

/*************************
 *
 *************************/

Data::Data(DataFile &df)
 : fname(NULL), idim(df.GetIdim()), odim(df.GetOdim()),
nb_totl(df.GetNbex()),
   preload(0), resampl_mode(0), resampl_seed(1234567890),
shuffle_mode(0),
   norm_mode(0),
   idx(-1), mem_inp(NULL), mem_trg(NULL), input(NULL), target(NULL)
{

  datafile.push_back(&df);
}

Data::~Data()
{
  if (preload) {
    delete [] mem_inp;
    if (odim>0) delete [] mem_trg;
  }
  for (vector<DataFile*>::iterator it = datafile.begin();
it!=datafile.end(); ++it)
      delete (*it);
  datafile.clear();
}

/*************************
 *
 *************************/
```

```
void Data::Shuffle()
{
  if (shuffle_mode < 1 || !preload) return;

  REAL      *inp = new REAL[idim];
  REAL      *trg = new REAL[odim];

  cout << " - shuffling data " << shuffle_mode << " times ...";
  cout.flush();
  for (int i=0; i<shuffle_mode*nb_totl; i++) {
    int i1 = (int) (nb_totl * drand48());
    int i2 = (int) (nb_totl * drand48());

    memcpy(inp, mem_inp + i1*idim, idim*sizeof(REAL));
    memcpy(mem_inp + i1*idim, mem_inp + i2*idim, idim*sizeof(REAL));
    memcpy(mem_inp + i2*idim, inp, idim*sizeof(REAL));

    if (odim>0) {
      memcpy(trg, mem_trg + i1*odim, odim*sizeof(REAL));
      memcpy(mem_trg + i1*odim, mem_trg + i2*odim, odim*sizeof(REAL));
      memcpy(mem_trg + i2*odim, trg, odim*sizeof(REAL));
    }

  }

  delete [] inp; delete [] trg;
  cout << " done" << endl;
}

//*************************
//
//

void Data::Preload()
{
  if (!preload || preload>DATA_PRELOAD_ONCE) return;
  if (preload == DATA_PRELOAD_ONCE) preload=DATA_PRELOAD_DONE;

  cout << " - loading all data into memory" << endl;

  int idx=0;
  for (vector<DataFile*>::iterator it = datafile.begin();
it!=datafile.end(); ++it) {
    (*it)->Rewind();
    int n = -1, maxn = (*it)->GetNbresampl();
//cout << "Resampl " << maxn << " examples from file into " <<  (*it)-
>input << endl;
    while (++n < maxn) {
      (*it)->Resampl();
//cout << "n: " << n << ", idx: " << (*it)->idx << endl;
      memcpy(mem_inp+idx*idim, (*it)->input, idim*sizeof(REAL));
      if (odim > 0) memcpy(mem_trg+idx*odim, (*it)->target_vect,
odim*sizeof(REAL));
      idx++;
    }
  }
```

```
  if (norm_mode & 1) {
    cout << " - normalizing input: subtract mean" << endl;
    for (int i=0; i<idim; i++) {
      int e;
      REAL m=0, *mptr;
      for (e=0, mptr=mem_inp+i; e<idx; e++, mptr+=idim) m+=*mptr;
      m = m/idx; // mean
      for (e=0, mptr=mem_inp+i; e<idx; e++, mptr+=idim) *mptr -= m;
    }
  }

  if (norm_mode & 2) {
    cout << " - normalizing input: divide by variance" << endl;
    for (int i=0; i<idim; i++) {
      int e;
      REAL m=0, m2=0, *mptr;
      for (e=0, mptr=mem_inp+i; e<idx; e++, mptr+=idim) { m+=*mptr;
m2+=*mptr * *mptr; }
      m = m/idx;  // mean
      m2 = m2/idx - m; // var = 1/n sum_i x_i^2  -  mu^2
      if (m2>0)
        for (e=0, mptr=mem_inp+i; e<idx; e++, mptr+=idim)
          *mptr = (*mptr - m) / m2;
    }
  }
#ifdef DEBUG
    for (int e=0; e<idx; e++) {
      for (int i=0; i<idim; i++) printf(" %5.2f",mem_inp[e*idim+i]);
printf("\n");
    }
#endif
}


/*************************
 *
 *************************/

void Data::Rewind()
{
  if (preload) {
       // clear all data, resample and shuffle again
    Preload();
    Shuffle();
  }
  else {
    for (vector<DataFile*>::iterator it = datafile.begin();
it!=datafile.end(); ++it) (*it)->Rewind();
  }
  idx = -1;
}

/*************************
 * Advance to next data
 *************************/

bool Data::Next()
```

```
{
  if (idx >= nb_totl-1) return false;
  idx++;

  if (preload) {
      // just advance to next data in memory
    input = &mem_inp[idx*idim];
    if (odim>0) target = &mem_trg[idx*odim];
//printf("DATA:"); for (int i =0; i<idim; i++) printf(" %5.2f",
input[i]); printf("\n");
    return true;
  }

  if (shuffle_mode > 0) {
      // resample in RANDOMLY SELECTED datafile until data was found
      // we are sure to find something since idx was checked before
    int df = (int) (drand48() * datafile.size());
//cout << " df=" << df << endl;
    datafile[df]->Resampl();
    input = datafile[df]->input;
    if (odim>0) target = datafile[df]->target_vect;
  }
  else {
      // resample SEQUENTIALLY all the data files
    static int df=0, i=-1, nbdf=datafile[df]->GetNbex();
    if (idx==0) {df = 0, i=-1, nbdf=datafile[df]->GetNbex(); }    //
(luint) this) is a hack to know when there was a global rewind
    if (++i >= nbdf) { df++; nbdf=datafile[df]->GetNbex(); i=-1; }
    if (df >= (int) datafile.size()) Error("internal error: no examples
left\n");
//printf("seq file: df=%d, i=%d\n", df,i);
    datafile[df]->Resampl();  //TODO: idx= ??
//cout << " got df=" << df << " idx="<<idx<<endl;
    input = datafile[df]->input;
    if (odim>0) target = datafile[df]->target_vect;
  }

  return true;
}
```

B.5 Data.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
```

```
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Data.h,v 1.8 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Data_h
#define _Data_h

#include <iostream>
#include <fstream>
#include <vector>
#include "Tools.h"
#include "DataFile.h"

// Names of information in files

extern const int   DATA_LINE_LEN;
extern const char* DATA_HEADER_TXT;
extern const int   DATA_HEADER_ID;
extern const char* DATA_PRELOAD;
extern const char* DATA_RESAMPL_MODE;
extern const char* DATA_RESAMPL_SEED;
extern const char* DATA_SHUFFLE_MODE;

/*
 * Strategie
 *  - there is one funtion Rewind() and Next() which should not be
overriden
 *  - they perform all the processing with preloading, shuffling, etc
 *  - the class specific processing is done in First() and Advance()
 */


class Data
{
protected:
  char *fname;
  int  idim, odim;      // dimensions
  int  nb_totl;         // number of examples
    // flags
  int preload;          //
  int resampl_mode;     //
  int resampl_seed;     //
  int shuffle_mode;     //
  int norm_mode;  // evtl. perform normalization; bits: 1=substract
mean, 2=devide by var.
    // data files
```

```
  vector<DataFile*>     datafile;
    // actual data
  int  idx;       // index of current example [0,nb-1]
  REAL *mem_inp;  // all the input data in memory
  REAL *mem_trg;  // all the output data in memory
    // local tools, only used when prelaod is activated
  void Preload(); // preload all data
  void Shuffle(); // shuffle in memory
public:
  Data(char *fname);
  Data(DataFile&);      // simplified version with one Datafile only
  ~Data();
    // access function to local variables
  char *GetFname() {return fname;}
  int GetIdim() {return idim;}
  int GetOdim() {return odim;}
  int GetNb() {return nb_totl;}
  int GetIdx() {if (idx<0) Error("DataNext() must be called before
GetIdx()"); return idx;};
    // the following two pointers are only valid after first DataNext()
!
  REAL *input;          // pointer to current inputs
  REAL *target;         // pointer to current target
  //REAL *GetData() {return val;}
    // main functions to access data
  void Rewind();  // rewind to first example, performs resmplaing,
shuffling etc if activated
  bool Next();          // advance to next example, return FALSE if at
end
};

#endif
```

B.6 DataAscii.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataAscii.cpp,v 1.8 2010/01/25 12:53:31 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "Data.h"
#include "DataAscii.h"

const char* DATA_FILE_ASCII="DataAscii";

DataAscii::DataAscii(ifstream &ifs) : DataFile::DataFile(ifs)
{

  dfs.open(fname,ios::in);
  CHECK_FILE(dfs,fname);
  char buf[DATA_LINE_LEN];
  dfs.getline(buf,DATA_LINE_LEN);
  sscanf(buf, "%d %d %d", &nbex, &idim, &odim);
  printf(" - %s: ASCII data with %d examples of dimension %d -> %d\n",
fname, nbex, idim, odim);


  if (idim>0) input = new REAL[idim];
  if (odim>0) target_vect = new REAL[odim];
}


/*************************
 *
 *************************/

DataAscii::~DataAscii()
{
  dfs.close();
  if (idim>0) delete [] input;
  if (odim>0) delete [] target_vect;
}


/*************************
 *
 *************************/

void DataAscii::Rewind()
{
  // dfs.seekg(0,ios::beg); HACK: does not work
  dfs.close();
  dfs.open(fname,ios::in);
  CHECK_FILE(dfs,fname);
  char buf[DATA_LINE_LEN];
  dfs.getline(buf,DATA_LINE_LEN);
```

```
}

/*************************
 *
 *************************/

bool DataAscii::Next()
{
  char line[DATA_LINE_LEN];
  dfs.getline(line, DATA_LINE_LEN);
  if (dfs.eof()) return false;
          else idx++;

    // parse input data
  char *lptr=line;
//cout << "\nLINE: " << line << endl;
  for (int i=0; i<idim; i++) {
//cout << "parse:" <<lptr<<"; ";
    while (*lptr==' ' || *lptr=='\t') lptr++;
    if (!*lptr) Error("incomplete input in ASCII datafile");
    if (sscanf(lptr, "%f", input+i)!=1) Error("parsing source in ASCII
datafile");
//cout << "got i[" <<i << "] " << input[i] << endl;
    while (*lptr!=' ' && *lptr!='\t' && *lptr!=0) lptr++;
  }

  if (odim<=0) return true;

    // parse target data
  for (int i=0; i<odim; i++) {
//cout << "parse:" <<lptr<<"; ";
    while (*lptr==' ' || *lptr=='\t') lptr++;
    if (!*lptr) Error("incomplete target in ASCII datafile");
    if (sscanf(lptr, "%f", target_vect+i)!=1) Error("parsing target in
ASCII datafile");
//cout << "got t[" <<i << "] " << target_vect[i] << endl;
    while (*lptr!=' ' && *lptr!='\t' && *lptr!=0) lptr++;
  }

  return true;
}
```

B.7 DataAscii.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
```

```
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataAscii.h,v 1.4 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _DataAscii_h
#define _DataAscii_h

#include <iostream>
#include <fstream>

#include "DataFile.h"

extern const char* DATA_FILE_ASCII;

class DataAscii : public DataFile
{
protected:
  ifstream dfs;
public:
  DataAscii(ifstream &ifs);
  virtual ~DataAscii();
  virtual void Rewind();
  virtual bool Next();
};

#endif
```

B.8 DataFile.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
```

```
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataFile.cpp,v 1.6 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
#include "Data.h"
#include "DataFile.h"

DataFile::DataFile(ifstream &ifs)
 : idim(0), odim(0), nbex(0), resampl_coeff(1.0), fname(NULL),
   idx(-1), input(NULL), target_vect(NULL)
{
  char p_fname[DATA_LINE_LEN];

  ifs >> p_fname >> resampl_coeff;
  if (resampl_coeff<=0 || resampl_coeff>1)
    Error("resampl coefficient must be in (0,1]\n");
  fname=strdup(p_fname);

  // memory allocation of input and target_vect should be done in
subclass
  // in function of the dimension and number of examples
}

DataFile::DataFile(char *p_fname, float p_rcoeff)
 : idim(0), odim(0), nbex(0), resampl_coeff(p_rcoeff),
fname(strdup(p_fname)),
   idx(-1), input(NULL), target_vect(NULL)
{

  // memory allocation of input and target_vect should be done in
subclass
  // in function of the dimension and number of examples
}

DataFile::~DataFile()
{
  if (fname) free(fname);
  // memory deallocation of input and target_vect should be done in
subclass

}
```

```
/*************************
 *
 *************************/

int DataFile::Info()
{
  int nbr=resampl_coeff*nbex;
  printf(" - %s  %6.4f * %9d = %9d\n", fname, resampl_coeff, nbex,
nbr);
  return nbr;
}

/*************************
 *
 *************************/

void DataFile::Rewind()
{
  Error("DataFile::Rewind() should be overriden");
}

//*************************
// read next data in File
// Return false if EOF

bool DataFile::Next()
{
  Error("DataFile::Next() should be overriden");
  return false;
}

//*************************
// generic resampling function using sequential file reads
// cycles sequentially through data until soemthing was found
// based on DataNext() which may be overriden by subclasses
// returns idx of current example

int DataFile::Resampl()
{
  bool ok=false;

  while (!ok) {
    if (!Next()) Rewind(); // TODO: deadlock if file empty
//cout << "Resampled: ";
//for (int i=0; i<idim; i++) cout << input[i] << " ";
    ok = (drand48() < resampl_coeff);
//cout << " ok=" << ok << endl;
  }

  return idx;
}
```

B.9 DataFile.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataFile.h,v 1.5 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _DataFile_h
#define _DataFile_h

#include <iostream>
#include <fstream>
#include <vector>
#include "Tools.h"

class DataFile {
protected:
  int idim, odim, nbex;
  float resampl_coeff;
    // internal handling of data
  char *fname;
public:
    // current data
  int  idx;
  REAL *input;          // current input data
  REAL *target_vect;    //         output data
  int  target_id; //         index of output [0..odim)
   // functions
  DataFile(ifstream &ifs);
  DataFile(char *, float =1.0);
  virtual ~DataFile();
   // access function
  int GetIdim() { return idim; }
  int GetOdim() { return odim; }
```

```
  int GetNbex() { return nbex; }
  int GetNbresampl() { return (int) (nbex*resampl_coeff); }
  float GetResampl() { return resampl_coeff; }
   // main interface
  virtual int Info();         // display line with info after loading
the data
  virtual void Rewind();      // rewind to first element
  virtual bool Next();        // advance to next data
  virtual int Resampl();      // resample another data (this may skip
some elements in the file)
};

#endif
```

B.10 DataNgramBin.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataNgramBin.cpp,v 1.11 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>

// system headers
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include "Tools.h"
#include "Data.h"
#include "DataNgramBin.h"
```

```
const char* DATA_FILE_NGRAMBIN="DataNgramBin";
const int DATA_NGRAM_IGN_BOS=1;
const int DATA_NGRAM_IGN_UNK=2;
const int DATA_NGRAM_IGN_UNKall=4;
const int DATA_NGRAM_IGN_EOS=8;     // TODO: not implemented
const int DATA_NGRAM_IGN_ALL=15;


//******************

void DataNgramBin::do_constructor_work()
{
    // parse header binary Ngram file
  fd=open(fname, O_RDONLY);
  if (fd<0) {
    perror(fname); Error();
  }
  read(fd, &nbl, sizeof(int));
  read(fd, &nbex, sizeof(int));
  read(fd, &vocsize, sizeof(int));
  int s;
  read(fd, &s, sizeof(int));
  if (s != sizeof(WordID)) {
    fprintf(stderr,"binary n-gram data uses %d bytes per index, but
this code is compiled for %d byte indices\n", s, (int) sizeof(WordID));
    Error();
  }
  read(fd, &bos, sizeof(WordID));
  read(fd, &eos, sizeof(WordID));
  read(fd, &unk, sizeof(WordID));
  printf(" - %s binary ngram file with %d words in %d lines, order=%d,
mode=%d\n", fname, nbex, nbl, order, mode);

  idim=order-1;
  odim=1;

  if (idim>0) {
    input = new REAL[idim];
    wid = new WordID[order];
    for (int i=0; i<order; i++) wid[i]=bos;
  }
  target_vect = new REAL[odim];

    // counting nbex to get true number of examples
  cout << "    counting ..."; cout.flush();
  int n=0;
  nbs=nbw=nbu=nbi=0;
  while (DataNgramBin::Next()) n++;
  printf(" %d %d-grams (%d unk, %d ignored)\n", n, order, nbu, nbi);
  if (n>nbex)
    Error("Number of counted examples is larger than information in
file header !?");
  nbex=n;  //
}

//******************
```

```
DataNgramBin::DataNgramBin(ifstream &ifs) : DataFile::DataFile(ifs),
  order(4), mode(0), nbw(0), nbs(0), nbu(0), nbi(0)
{
    // DataNgramBin <file_name> <resampl_coeff> <order> [flags]
    // parse addtl params
  ifs >> order >> mode;
  if (order<2 || order>9)
    Error("order must be in [2,9]\n");
  if (mode<0 || mode>DATA_NGRAM_IGN_ALL)
    Error("wrong value of DataNgramBin mode\n");

  do_constructor_work();
}

//*******************

DataNgramBin::DataNgramBin(char *p_fname, float p_rcoeff, int p_order,
int p_mode)
  : DataFile::DataFile(p_fname, p_rcoeff),
  order(4), mode(p_mode), nbw(0), nbs(0), nbu(0), nbi(0)
{

  do_constructor_work();
    // skip counting for efficieny reasons
  nbw=nbex;       // this should be an upper bound on the number of n-
grams
}

//*******************

DataNgramBin::~DataNgramBin()
{

  close(fd);
  if (idim>0) {
    delete [] wid;
    delete [] input;
  }
  delete [] target_vect;
}


//*******************

bool DataNgramBin::Next()
{
  bool ok=false;
  int i;

   // we may need to skip some n-grams in function of the flags
  while (!ok) {

      // read from file into, return if EOF
    WordID w;
    if (read(fd, &w, sizeof(w)) != sizeof(w)) return false;
//printf("read: %d\n",w);
```

```
      // shift previous order
    for (i=1; i<order; i++) wid[i-1]=wid[i];
    wid[order-1]=w;
//printf(" wid: %d %d %d\n",wid[0],wid[1],wid[2]);

     // update statistics
    if (w == bos) ; /* nothing to count */
    else if (w == eos) nbs++;
    else if (w == unk) nbu++;
    else nbw++;

     // check if n-gram is valid according to the selected mode

    if (w == bos) {
        // new BOS, initialize the whole order to BOS
        // (it will be shifted away)
      for (i=0; i<order; i++) wid[i] =  bos;
//printf("skip [new bos]\n");
      continue;
    }

    if (mode & DATA_NGRAM_IGN_UNK) {
        // ignore n-grams with <UNK> at last position
      if (w == unk) {
        nbi++;
//printf("skip [predict unk]\n");
        continue;
      }
    }

    if (mode & DATA_NGRAM_IGN_UNKall) {
        // ignore n-grams that contain <UNK> anywhere
      for (i=0; i<order-1; i++) {
        if (wid[i] == unk) {
          nbi++;
//printf("skip [any unk]\n");
          break;
        }
      }
      if (i < order-1) continue;
    }

    if (mode & DATA_NGRAM_IGN_BOS) {
        // ignore n-grams that contain <BOS> elsewhere than at 1st
position
      for (i=1; i<order; i++)
        if (wid[i] == bos) {
          nbi++;
//printf("skip [bos]\n");
          break;
      }
      if (i < order) continue;
    }

      /* standard mode */
    ok=true;
  } // of while (!ok)
```

```
//printf("keep: %d %d %d\n",wid[0],wid[1],wid[2]);
  for (i=0; i<order-1; i++) input[i] = (REAL) wid[i];       // careful:
we cast to float which may give
  target_vect[0] = (int) wid[i];                       // rounding
problems of the integers
  target_id = (int) wid[i];

  idx++;
  return true;
}

/********************
 *
 ********************/


int DataNgramBin::Info()
{
  return DataFile::Info();
  //int nbr=resampl_coeff*nbex;
  //printf(" - %s  %6.4f * %9d = %9d [ngram order=%d, mode=%d, unk=%d,
bos=%d, eos=%d]\n", fname, resampl_coeff, nbex, nbr, order, mode, unk,
bos, eos);
  //return nbr;
}

void DataNgramBin::Rewind()
{
  lseek(fd,
sizeof(nbl)+sizeof(nbex)+sizeof(vocsize)+sizeof(int)+3*sizeof(WordID),
SEEK_SET);
  idx=-1;
}
```

B.11 DataNgramBin.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
```

```
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataNgramBin.h,v 1.4 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _DataNgramBin_h
#define _DataNgramBin_h

#include <iostream>
#include <fstream>

#include "DataFile.h"

extern const char* DATA_FILE_NGRAMBIN;

typedef int WordID;

// Syntax of a line in data description:
// DataNgramBin <file_name> <resampl_coeff> <order> [flags]
//  u: skip n-grams with <unk> at the right most position
//  U: skip n-grams with <unk> anywhere
//  b: skip n-grams with <s> elsewhere than at the left most position
//  e: skip n-grams with </s> elsewhere than at the right most position

class DataNgramBin : public DataFile
{
private:
  void do_constructor_work();
protected:
  int fd;         // UNIX style binary file
  int vocsize;          // vocab size (including <s>, </s> and <unk>)
  int order;            // order of the ngrams
  int mode;       // see above for possible flags
  WordID *wid;          // whole n-gram context
  WordID bos, eos, unk; // word ids of special symbols
    // stats (in addition to nbex in mother class)
  int  nbl, nbw, nbs, nbu;// lines, words, sentences, unks
  int  nbi;       // ignored n-grams
public:
  DataNgramBin(ifstream &ifs);
  DataNgramBin(char*, float =1.0, int =4, int =3);
  virtual ~DataNgramBin();
  virtual int Info();
  virtual bool Next();
  virtual void Rewind();
};

#endif
```

B.12 ErrFct.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFct.cpp,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "ErrFct.h"

ErrFct::ErrFct (Mach &mach)
 : dim(mach.GetOdim()), bsize(mach.GetBsize()),
   output(mach.GetDataOut()), target(NULL), grad(new REAL[dim*bsize])
{
//cerr << "Constructor ErrFct: alloc gradient of size " << dim << endl;
}

//*****************************************************************
****************

REAL ErrFct::CalcValue(int eff_bsize) { return 0; }

REAL ErrFct::CalcGrad(int eff_bsize) {
  if (eff_bsize<=0) eff_bsize=bsize;
  for (int i=0; i<dim*eff_bsize; i++) grad[i]=0.0;
  return 0;
}
```

B.13 ErrFct.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFct.h,v 1.6 2010/01/25 12:27:07 schwenk Exp $
 *
 * Class definiton of a general error funtion
 */

#ifndef _ErrFct_h
#define _ErrFct_h

#include <iostream>
#include "Tools.h"
#include "Mach.h"
#include "Data.h"


class ErrFct
{
private:
protected:
  int dim;              // output dimension of machine
  int bsize;
  REAL  *output;        // pointer to output data (stored in machine)
  REAL  *target;        // pointer to target data (stored in trainer)
  REAL  *grad;               // calculated gradient (stored in this
class)
public:
  ErrFct(Mach&);
  virtual ~ErrFct() { delete [] grad; }
  void SetOutput(REAL *p_output) {output=p_output; }
  void SetTarget(REAL *p_target) {target=p_target; }
  REAL *GetGrad() {return grad; };
```

```
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                 // calculate NEGATIF
gradient of error function
};

#endif
```

B.14 ErrFctCrossEnt.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctCrossEnt.cpp,v 1.4 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "ErrFctCrossEnt.h"

//*********************************************************************
*************
// E = sum_i d_i ln o_i
REAL ErrFctCrossEnt::CalcValue(int eff_bsize) {
  REAL      *optr=output;
  REAL      *tptr=target;
  double err=0.0;

  if (eff_bsize<=0) eff_bsize=bsize;
  for (int i=0; i<eff_bsize*dim; i++) {
      err += *tptr++ * log(*optr++);
```

```
  }
  return (REAL) err/dim/eff_bsize;
}

// dE / do_i = d_i / t_i
REAL ErrFctCrossEnt::CalcGrad(int eff_bsize) {
  REAL      *optr=output;
  REAL      *tptr=target;
  REAL      *gptr=grad;
  REAL err=0.0;

  if (eff_bsize<=0) eff_bsize=bsize;
  for (int i=0; i<eff_bsize*dim; i++) {
    *gptr++ = (*optr == 0) ? 0 : *tptr / *optr; // TODO
    err += *tptr++ * log(*optr++);
  }
  return (REAL) err/dim/eff_bsize;
}
```

B.15 ErrFctCrossEnt.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctCrossEnt.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 *
 * Class definiton of cross entropy error function
 *   E = sum_i  d_i * ln o_i
 *   dE/do_k = d_k / o_k   for o_k <> 0
 * This is usually used with softmax outputs
 */

#ifndef _ErrFctCrossEnt_h
#define _ErrFctCrossEnt_h
```

```
#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctCrossEnt : public ErrFct
{
public:
  ErrFctCrossEnt(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                // calculate NEGATIF
gradient of error function
};

#endif
```

B.16 ErrFctCrossEntNgram.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctCrossEntNgram.cpp,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "ErrFctCrossEntNgram.h"
```

```
ErrFctCrossEntNgram::ErrFctCrossEntNgram (Mach &mach)
 : dim(mach.GetOdim()), bsize(mach.GetBsize()),
   output(mach.GetDataOut()), target(NULL), grad(new REAL[bsize])
{
//cerr << "Constructor ErrFct: alloc gradient of size " << dim << endl;
}


//*****************************************************************
*************
// E = sum_i d_i ln o_i
REAL ErrFctCrossEntNgram::CalcValue(int eff_bsize) {
  REAL      *optr=output;
  REAL      *tptr=target;
  double err=0.0;

  if (eff_bsize<=0) eff_bsize=bsize;
  for (int i=0; i<eff_bsize*dim; i++) {
      err += *tptr++ * log(*optr++);
  }
  return (REAL) err/dim/eff_bsize;
}

// dE / do_i = d_i / t_i
REAL ErrFctCrossEntNgram::CalcGrad(int eff_bsize) {
  REAL      *optr=output;
  REAL      *tptr=target;
  REAL      *gptr=grad;
  REAL err=0.0;

  if (eff_bsize<=0) eff_bsize=bsize;
  for (int i=0; i<eff_bsize*dim; i++) {
    *gptr++ = (*optr == 0) ? 0 : *tptr / *optr; // TODO
    err += *tptr++ * log(*optr++);
  }
  return (REAL) err/dim/eff_bsize;
}
```

B.17 ErrFctCrossEntNgram.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
```

```
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctCrossEntNgram.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 * Class definiton of cross entropy error function
 * Spezial version for NNs that predict words
 *  - the NN has a large output dimension (vocsize or limited to
shorlist)
 *  - the data has one dimensional targets that are taken as index into
 *    the word list
 *  - therfore the target vector is binary: 1 at the position of the to
predicted
 *    word, 0 elsewhere
 *
 *   E = sum_i  d_i * ln o_i
 *   dE/do_k = d_k / o_k   for o_k <> 0
 * This is usually used with softmax outputs
 */

#ifndef _ErrFctCrossEnt_h
#define _ErrFctCrossEnt_h

#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctCrossEnt : public ErrFct
{
private:
  int voc_size;         //
                 // the private var "dim" is set to 1
public:
  ErrFctCrossEnt(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                // calculate NEGATIF
gradient of error function
};

#endif
```

B.18 ErrFctMCE.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
```

```
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctMCE.cpp,v 1.4 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "ErrFctMCE.h"

//**********************************************************************
*****************

REAL ErrFctMCE::CalcValue(int eff_bsize) {
  REAL      *optr=output;
  REAL      *tptr=target;
  int nb_err=0;

  if (eff_bsize<=0) eff_bsize=bsize;
  for (int b=0; b<eff_bsize; b++) {
    REAL omax=optr[0], tmax=tptr[0];
    int oidx=0, tidx=0;
    for (int i=0; i<dim; i++) {
      if (*optr > omax) {omax=*optr; oidx=i;}
      if (*tptr > tmax) {tmax=*tptr; tidx=i;}
//printf("%f %f\n", *optr, *tptr);
      optr++; tptr++;
    }
    if (oidx!=tidx) nb_err++;
//printf("b=%d, oidx=%d, tidx=%d, err=%d\n", b, oidx, tidx, nb_err);
  }

  return (REAL) nb_err/eff_bsize;
}
```

```
REAL ErrFctMCE::CalcGrad(int eff_bsize) {
  REAL      *optr=output;
  REAL      *tptr=target;
  REAL      *gptr=grad;
  int nb_err=0;

  if (eff_bsize<=0) eff_bsize=bsize;

  for (int b=0; b<eff_bsize; b++) {
    REAL omax=optr[0], tmax=tptr[0];
    int oidx=0, tidx=0;
    for (int i=0; i<dim; i++) {
      if (*optr > omax) {omax=*optr; oidx=i;}
      if (*tptr > tmax) {tmax=*tptr; tidx=i;}
      *gptr++ = -(*optr++ - *tptr++);
    }
    if (oidx!=tidx) nb_err++;
  }
  return (REAL) nb_err/eff_bsize;
}
```

B.19 ErrFctMCE.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctMCE.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 *
 * Class definiton of ``mean classification error'' function (MCE)
 *   we use MSE for training, but the value of the error function is
 *   the percentage of wrongly classified examples
 */

#ifndef _ErrFctMCE_h
```

```
#define _ErrFctMCE_h

#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctMCE : public ErrFct
{
public:
  ErrFctMCE(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                 // calculate NEGATIF
gradient of error function
};

#endif
```

B.20 ErrFctMSE.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctMSE.cpp,v 1.4 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "ErrFctMSE.h"
```

```
//*****************************************************************
*****************

REAL ErrFctMSE::CalcValue(int eff_bsize) {
  REAL mse=0.0, val;
  REAL      *optr=output;
  REAL      *tptr=target;

  if (eff_bsize<=0) eff_bsize=bsize;
  for (int i=0; i<dim*eff_bsize; i++) {
//printf("o=%f t=%f\n", *optr,*tptr);
    val = *optr++ - *tptr++;
    mse += val*val;
  }
//printf("MSE %f %d %d\n", mse, dim, eff_bsize);
  return mse/dim/eff_bsize/2;
}

REAL ErrFctMSE::CalcGrad(int eff_bsize) {
  REAL mse=0.0, val;
  REAL      *optr=output;
  REAL      *tptr=target;
  REAL      *gptr=grad;

//cout << "MSE" << eff_bsize << endl;
  if (eff_bsize<=0) eff_bsize=bsize;
  for (int i=0; i<dim*eff_bsize; i++) {
//printf(" %d: o=%f, t=%f\n", i, *optr, *tptr);
    val = *optr++ - *tptr++;
    *gptr++ = - val;
    mse += val*val;
  }
//cout << " avg=" << mse/dim/eff_bsize << endl;
  return mse/dim/eff_bsize/2;
}
```

B.21 ErrFctMSE.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
```

```
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctMSE.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 *
 * Class definiton of mean squared error error function (MSE)
 *   E = sum_i  (o_i - d_i)^2
 *   dE/do_k = 2 (o_k - d_k)
 */

#ifndef _ErrFctMSE_h
#define _ErrFctMSE_h

#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctMSE : public ErrFct
{
public:
  ErrFctMSE(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                // calculate NEGATIF
gradient of error function
};

#endif
```

B.22 ErrFctSoftmCrossEntNgram.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
```

```
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctSoftmCrossEntNgram.cpp,v 1.5 2010/01/25 12:27:07 schwenk
Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "ErrFctSoftmCrossEntNgram.h"


//*********************************************************************
*************
// E = log(sum_i d_i ln o_i)
//   = ln o_t     where t is the target index
//   output: dimension voc_size
//   target: dimension 1 with values [0,voc_size[
// We also take the log since this can't be done later if bsize>1

REAL ErrFctSoftmCrossEntNgram::CalcValue(int eff_bsize) {
  REAL *optr=output;
  REAL *tptr=target;
  double err=0.0;

  if (eff_bsize<=0) eff_bsize=bsize;
  for (int b=0; b<eff_bsize; b++) {
    if (*tptr<0 || *tptr>=dim) {
      printf("ErrFctSoftmCrossEntNgram::CalcValue(): target out of
bounds (%d) must be in [0,%d[\n",(uint)*tptr,dim);
      Error();
    }
//printf("b=%d, tidx=%f, out=%f\n", b, *tptr, optr[(uint) *tptr]);
    err += log(optr[(uint) *tptr++]);
//printf("err=%f\n",err);
    optr += dim;
  }
  return (REAL) err; // TODO: normalize ?
}

// We include here the derivation of the softmax outputs since we have
//   dE/da_k = sum_i dE/do_i do_i/da_k
// Due to the sum, dE/do_i and do_i/da_k can't be calculated separately
// dE/do_i = d_i/o_i
// do_i/da_k = o_i (kronecker_ik - o_k)
//  -> dE/da_k = sum_i d_i/o_i * o_i (kronecker_ik - o_k)
//             = sum_i d_i (kronecker_ik - o_k)
//             = (kronecker_tk - o_k)       since d_i=0 for i!=t
REAL ErrFctSoftmCrossEntNgram::CalcGrad(int eff_bsize) {
  REAL *optr=output;
```

```
  REAL *tptr=target;
  REAL *gptr=grad;
  REAL err=0.0;
  uint      tidx;

  if (eff_bsize<=0) eff_bsize=bsize;
  for (int b=0; b<eff_bsize; b++) {
    for (int i=0; i<dim; i++) gptr[i] = -optr[i];
    tidx=(uint) *tptr++;
    if (tidx<0 || tidx>=(uint) dim) {
      printf("ErrFctSoftmCrossEntNgram::CalcGrad(): target out of
bounds (%d) must be in [0,%d[\n",tidx,dim);
      Error();
    }
    err += log(optr[tidx]);
    gptr[tidx] += 1.0;
    gptr+=dim; optr+=dim;
  }
  return (REAL) err; // TODO: normalize ?
}
```

B.23 ErrFctSoftmCrossEntNgram.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctSoftmCrossEntNgram.h,v 1.4 2010/01/25 12:27:07 schwenk
Exp $
 *
 * Class definiton of cross entropy error function
 * Spezial version for NNs that predict words
 *  - the NN has a large output dimension (vocsize or limited to
shorlist)
 *  - the data has one dimensional targets that are taken as index into
```

```
 *    the word list
 *  - therefore the target vector is binary: 1 at the position of the
to predicted
 *    word, 0 elsewhere
 *
 *   E = sum_i  d_i * ln o_i
 *   dE/do_k = d_k / o_k   for o_k <> 0
 * This is usually used with softmax outputs
 */

#ifndef _ErrFctSoftmCrossEnt_h
#define _ErrFctSoftmCrossEnt_h

#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctSoftmCrossEntNgram : public ErrFct
{
private:
  int voc_size;         //
                 // the private var "dim" is set to 1
public:
  ErrFctSoftmCrossEntNgram(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);    // Calculate value of error
function
  virtual REAL CalcGrad(int=0);           // calculate NEGATIF gradient
of error function
};

#endif
```

B.24 Eval.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Eval.h,v 1.2 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Eval_h
#define _Eval_h

#include <iostream>
#include "Tools.h"
#include "Mach.h"
#include "Data.h"


class Eval
{
private:
protected:
  Mach  &mach;                // network to evaluate
  int idim, odim, bsize;      // copied here for faster access
    // buffer to store bsize examples
  REAL  *buf_input;
  REAL  *buf_target;
public:
  Eval(Mach&, int=16384);
  virtual ~Eval();
  virtual void Data(Data &data, int* = NULL);   // evaluate on existing
data
  virtual void BlockEval(WordId &wid, int order, float *p, int n);
  virtual void BlockFinish();
};

#endif
```

B.25 EvalNgramBin.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
```

```
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: EvalNgramBin.cpp,v 1.4 2010/01/26 19:37:22 schwenk Exp $
 */

using namespace std;

#include "Tools.h"
#include "EvalNgramBin.h"

#include <algorithm>

EvalNgramBin::EvalNgramBin(Mach &p_mach, uint p_max_req)
 : mach(p_mach), max_req(p_max_req)
{

  idim=mach.GetIdim(); odim=mach.GetOdim(); bsize=mach.GetBsize();

  if (odim < 16) {
    fprintf(stderr,"EvalNgramBin: output dimension of the machine is
suspiciously small (%d)\n", odim);
    Error();
  }

  buf_input = new REAL[idim*bsize];
  mach.SetDataIn(buf_input);
}

EvalNgramBin::~EvalNgramBin()
{
  for (vector<NgramReq*>::iterator it=req.begin(); it<req.end(); ++it)
delete *it;
  delete [] buf_input;
}

REAL EvalNgramBin::Eval(WordID *wid, int order, float *p)
{
  if (order-1 != idim) {
    fprintf(stderr,"EvalNgramBin::Eval(): requested context size (%d)
does not match input dimension of neural network (%d)\n", order-1,
idim);
    Error();
  }
  int oidx= wid[order-1];
  if (oidx<0 || oidx>=odim) {
    fprintf(stderr,"EvalNgramBin::Eval(): wrong index of the predicted
word (%d), should be in [0,%d[\n", oidx, odim);
    Error();
  }

  for (int i=0; i<order-1; i++) buf_input[i]=(REAL) wid[i];
#ifdef DEBUG
```

```
  for (int i=0; i<order-1; i++) printf(" %d", wid[i]);
  printf(" -> %d\n", oidx);
#endif
  mach.Forw(1);

  if (p) *p=mach.GetDataOut()[oidx];

  return mach.GetDataOut()[oidx];
}

void EvalNgramBin::BlockEval(WordID *wid, int order, float *p)
{
  req.push_back(new NgramReq(wid, order, p));
  if (req.size()>=max_req) BlockFinish();
}

void EvalNgramBin::BlockFinish()
{
#ifdef DEBUG
  for (vector<NgramReq*>::iterator it=req.begin(); it<req.end(); ++it)
    (*it)->display();
#endif
  //sort(req.begin(),req.end());  // use operator < of Ngramreq
  sort(req.begin(),req.begin()+req.size()-1,NgramReq::Compare);
  //qsort(&req[0], req.size(), sizeof(req[0]), NgramReq::Compare);
#ifdef DEBUG
  for (int i=0; i<req.size(); i++) {
    printf("buf %d:", i); req[i]->display();
  }
#endif
  req.clear();
}
```

B.26 EvalNgramBin.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: EvalNgramBin.h,v 1.3 2010/01/26 19:37:22 schwenk Exp $
 */

#ifndef _EvalNgramBin_h
#define _EvalNgramBin_h

#include <iostream>
#include "Tools.h"
#include "Mach.h"
#include "DataNgramBin.h"

//
// helper class to store and compare one ngram LM request
//
class NgramReq {
  int ctxt_len;
  WordID *ctxt, wpred;
  float *res_ptr;
public:
  NgramReq(WordID *wid, int order, float *adrP)
    : ctxt_len(order-1), ctxt(new WordID[ctxt_len]),
wpred(wid[ctxt_len]), res_ptr(adrP)
    { for (int i=0; i<ctxt_len; i++) ctxt[i]=wid[i]; }
  ~NgramReq() {delete [] ctxt; }
   static bool Compare(NgramReq* n1, NgramReq *n2)
   { return true;
     for (int i=0; i<n1->ctxt_len; i++) {
       if (n1->ctxt[i] < n2->ctxt[i]) return true;
       if (n1->ctxt[i] > n2->ctxt[i]) return false;
     }
     return true; // both are equal
   }
   void display() {
     for (int c=0; c<ctxt_len; c++) printf(" %d", ctxt[c]);
     printf(" -> %d\n", wpred);
   }
};




class EvalNgramBin
{
private:
protected:
  Mach  &mach;                // network to evaluate
  int idim, odim, bsize;      // copied here for faster access
    // buffer to store bsize examples
  REAL  *buf_input;
    // buffers for block operations
  vector<NgramReq*> req;
```

```
  uint      max_req;    // max number of request cumulated before we
perform them in a block
public:
  EvalNgramBin(Mach&, uint=128);          // spezify one machine
  //EvalNgramBin(string, int=16384);            // spezify multiple
ipol machines
  virtual ~EvalNgramBin();
  //virtual void Data(Data &data, int* = NULL); // evaluate on existing
data
  virtual REAL Eval(WordID*, int, float* = NULL);     // get prob for 1
n-gram only
  virtual void BlockEval(WordID*, int, float*);
  virtual void BlockFinish();
};

#endif
```

B.27 Hypo.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Hypo.cpp,v 1.5 2010/01/28 09:27:12 schwenk Exp $
 */


#include "Hypo.h"
#include <iostream>

Hypo::Hypo()
{
  //cerr << "Hypo: constructor called" << endl;
}
```

```
Hypo::~Hypo()
{
  //cerr << "Hypo: destructor called" << endl;
}

void Hypo::Write(outputfilestream &outf)
{
  outf << id << NBEST_DELIM2 << trg << NBEST_DELIM2;
  for (vector<float>::iterator i = f.begin(); i != f.end(); i++)
    outf << (*i) << " ";
  outf << NBEST_DELIM << " " << s << endl;

}

float Hypo::CalcGlobal(Weights &w)
{
  //cerr << " HYP: calc global" << endl;
  uint sz=w.val.size();
  if (sz<f.size()) {
    cerr << " - NOTE: padding weight vector with " << f.size()-sz << "
zeros" << endl;
    w.val.resize(f.size());
    for (uint i=sz; i<w.val.size(); i++) w.val[i]=0;
  }

  s=0;
  for (uint i=0; i<f.size(); i++) {
    //cerr << "i=" << i << ", " << w.val[i] << ", " << f[i] << endl;
    s+=w.val[i]*f[i];
  }
  //cerr << "s=" << s << endl;
  return s;
}

// this is actually a "greater than" since we want to sort in
descending order
bool Hypo::operator< (const Hypo &h2) const {
  return (this->s > h2.s);
}
```

B.28 Hypo.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
```

```
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Hypo.h,v 1.6 2010/01/25 12:27:07 schwenk Exp $
 *
 * Basic functions to process one hypothesis
 */


#ifndef _HYPO_H_
#define _HYPO_H_

using namespace std;

#include <iostream>
#include <fstream>
#include <string>
#include <vector>

#include "Toolsgz.h"

#define NBEST_DELIM "|||"
#define NBEST_DELIM2 " ||| "

class Hypo {
protected:
  int id;
  string trg;  // translation
  vector<float> f;  // feature function scores
  float     s;  // global score
  // segmentation
public:
  Hypo();
  Hypo(int p_id,string &p_trg, vector<float> &p_f, float p_s) :
id(p_id),trg(p_trg),f(p_f),s(p_s) {};
  ~Hypo();
  float CalcGlobal(Weights&);
  void AddID(int o) {id+=o;};
  void Write(outputfilestream&);
  bool operator< (const Hypo&) const;
  // bool CompareLikelihoods (const Hypo&, const Hypo&) const;
  void SetFeature(float val, const int pos) {if(pos>0) f[pos-1]=val;
else f.push_back(val); };
  const char *GetCstr() {return trg.c_str(); };
};

#endif
```

B.29 Mach.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Mach.cpp,v 1.13 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "Mach.h"
#include "MachTab.h"
#include "MachTabShared.h"
#include "MachLin.h"
#include "MachSig.h"
#include "MachTanh.h"
#include "MachSoftmax.h"
#include "MachSeq.h"
#include "MachPar.h"

void Mach::do_alloc()
{
  if (odim*bsize>0) {
    data_out=::new REAL[odim*bsize];
    if (!data_out) Error ("can't allocate memory for data_out");
  }
  else data_out=NULL;
  data_in=NULL; // (luint) this) should be set later by SetDataIn()
  if (idim*bsize>0) {
    grad_in=::new REAL[idim*bsize];
    if (!grad_in) Error ("can't allocate memory for grad_in");
  }
  else grad_in=NULL;
```

```
  grad_out=NULL; // (luint) this) should be set later by SetGradOut()
}

Mach::Mach(const int p_idim, const int p_odim, const int p_bsize, const
int p_nbfw, const int p_nbbw)
 : idim(p_idim), odim(p_odim), bsize(p_bsize), nb_forw(p_nbfw),
nb_backw(p_nbbw)
{
  do_alloc();
}

Mach::~Mach()
{
  if (data_out) delete [] data_out;
  if (grad_in) delete [] grad_in;
}

//-----------------------------------------------
// File output
//-----------------------------------------------

void Mach::WriteParams(ofstream &of) {
    // write machine specific params
  of.write((char*) &nb_forw, sizeof(int));
  of.write((char*) &nb_backw, sizeof(int));
}

void Mach::WriteData(ofstream &of) {
  const int i=0, s=sizeof(REAL);
  of.write((char*) &i, sizeof(int));
  of.write((char*) &s, sizeof(int));
}

void Mach::Write(ofstream &of)
{
  char header[file_header_size];
  for (int i=0; i<file_header_size; i++) header[i]=' ';
  sprintf(header,"%s %d",file_header_name, file_header_version);
  of.write(header,file_header_size);
  of.write((char*) &idim, sizeof(int));
  of.write((char*) &odim, sizeof(int));
  of.write((char*) &bsize, sizeof(int));
  int mtype=GetMType();
  of.write((char*) &mtype, sizeof(int));
  WriteParams(of);
  WriteData(of);
}

//-----------------------------------------------
// File input
//-----------------------------------------------


void Mach::ReadParams(ifstream &inpf, bool with_alloc)
{
  inpf.read((char*) &nb_forw, sizeof(int));
  inpf.read((char*) &nb_backw, sizeof(int));
```

```
}

void Mach::ReadData(ifstream &inpf, size_t s)
{
  // there is nothing to read
}

Mach *Mach::Read(ifstream &inpf )
{
  char header[file_header_size], h[file_header_size];
  int v;

  inpf.read(header,file_header_size);
  if (sscanf(header,"%s %d",h,&v) != 2) {
    fprintf(stderr,"format of machine file not recognised: %s",
header);
    Error();
  }
  if (strcmp(h,file_header_name)) {
    fprintf(stderr, "unsupported file type (%s), expected '%s'\n", h,
file_header_name);
    Error();
  }
  switch (file_header_version) {
    case file_header_version: break;
    default:
      fprintf(stderr,"unsupported version of machine file (%d)\n",v);
      Error();
  }

    // read idim, odim, bsize
  int f_idim, f_odim, f_bsize;
  inpf.read((char*) &f_idim, sizeof(int));
  inpf.read((char*) &f_odim, sizeof(int));
  inpf.read((char*) &f_bsize, sizeof(int));

   // read and parse machine type
  int mtype;
  Mach *m;
  inpf.read((char*) &mtype, sizeof(int));
  switch (mtype) {
    case file_header_mtype_base: m = new Mach(f_idim,f_odim,f_bsize);
break;
    case file_header_mtype_tab: m = new
MachTab(NULL,f_idim,f_odim,f_bsize,0,0); break;
    case file_header_mtype_lin: m = new MachLin(f_idim,f_odim,f_bsize);
break;
    case file_header_mtype_sig: m = new MachSig(f_idim,f_odim,f_bsize);
break;
    case file_header_mtype_tanh: m = new
MachTanh(f_idim,f_odim,f_bsize); break;
    case file_header_mtype_softmax: m = new
MachSoftmax(f_idim,f_odim,f_bsize); break;
    case file_header_mtype_multi: m = new MachMulti(); break;
    case file_header_mtype_mseq: m = new MachSeq(); break;
    //case file_header_mtype_mstack: m = new MachStack; break;
    case file_header_mtype_mpar: m = new MachPar(); break;
```

```
    default:
      fprintf(stderr,"unknown machine type in file (%d)\n", mtype);
      Error();
  }

    // read rest of (machine specific) params
  m->ReadParams(inpf);

  int s;
  inpf.read((char*) &s,sizeof(int));  // number of elements
  inpf.read((char*) &v,sizeof(int));  // size in bytes of each element
  if (v != sizeof(REAL)) {
    fprintf(stderr, "binary data on file uses %d bytes while the
current code is compiled for %lu bytes\n", v, sizeof(REAL));
    Error();
  }
  m->ReadData(inpf, s);
  // TODO: check EOF

  return m;
}

//-------------------------------------------------
// Tools
//-------------------------------------------------

void Mach::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << " - dimensions: in=" << idim << ", out=" << odim << endl;
    cout << " - number of parallel examples=" << bsize << endl;
    cout << " - number of passes: " << nb_forw << "/" << nb_backw <<
endl;
  }
  else {
    printf("%sMach %d-%d, bs=%d, passes=%d/%d\n", txt, idim, odim,
bsize, nb_forw, nb_backw);
  }
}

//-------------------------------------------------
// Training
//-------------------------------------------------

void Mach::Forw(int eff_bsize)
{
  if (!data_in)
    Error("Mach::Forw(): input data is not set");
  if (idim!=odim)
    Error("Mach::Forw(): call to default Forw() function with different
dimensions");
  if (eff_bsize<=0) eff_bsize=bsize;
  memcpy(data_out, data_in, eff_bsize*idim*sizeof(REAL));
  nb_forw += eff_bsize;
}

void Mach::Backw (const float lrate, const float wdecay, int eff_bsize)
```

```
{
  if (!grad_out)
    Error("Mach::Backw(): output gradient is not set");
  if (idim!=odim)
    Error("Mach::Backw(): call to default Train() function with
different dimensions");
  if (eff_bsize<=0) eff_bsize=bsize;
  memcpy(grad_in, grad_out, eff_bsize*idim*sizeof(REAL));
  nb_backw += eff_bsize;
}
```

B.30 Mach.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Mach.h,v 1.11 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Machine_h
#define _Machine_h

#include <iostream>
#include <fstream>
#include "Tools.h"

#define BLAS      // use fast BLAS code, for instance with Intel's MKL
library

// list of all known machine types,
// this is needed for the general file read function

#define file_header_name "HPerf"
#define file_header_version 1
```

```
#define file_header_size 16

#define file_header_mtype_base            0
#define file_header_mtype_tab       1
#define file_header_mtype_tabsh           2
#define file_header_mtype_lin       3
#define file_header_mtype_sig       4
#define file_header_mtype_tanh            5
#define file_header_mtype_softmax   6
#define file_header_mtype_stab            7
#define file_header_mtype_multi           16
#define file_header_mtype_mseq            17
#define file_header_mtype_mstack    18
#define file_header_mtype_mpar            19

class Mach
{
private:
  void do_alloc();      // perform allocation of dynamic data
structures
protected:
  int  idim, odim;            // input and output dimension
  int  bsize;                 // block size (nb of example used in
parallel)
  int  nb_forw;               // nb of forward examples processed
  int  nb_backw;        // nb of backward examples processed
  REAL *data_in;        // input data (pointer)
  REAL *data_out;       // output data (allocated by machine)
  REAL *grad_in;        // input gradients (allocated by machine)
  REAL *grad_out;       // output gradients (pointer)
  // File I/O, the following functions can be overloaded by subclass
  // the main functions Read() and Write() should not be modified !
  virtual void ReadParams(ifstream&, bool=true); // read all params
  virtual void ReadData(ifstream&, size_t); // read binary data
  virtual void WriteParams(ofstream&); // write all params
  virtual void WriteData(ofstream&); // write binary data
public:
  Mach(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~Mach();
    // Tools
  virtual int GetMType() {return file_header_mtype_base;};  // get type
of machine
  virtual int GetIdim() {return idim;}
  int GetOdim() {return odim;}
  int GetBsize() {return bsize;}
  void SetBsize(int bs) {
    if (bs<1) Error("wrong value in SetBsize()"); else bsize=bs; }
  int GetNbForw() {return nb_forw;}
  int GetNbBackw() {return nb_backw;}
  virtual REAL* GetDataIn() {return data_in;} // return pointer on
input data for chaining
  virtual REAL* GetDataOut() {return data_out;} // return pointer on
output data for chaining
  virtual REAL* GetGradIn() {return grad_in;} // return pointer on
input gradient for chaining
```

```
  virtual REAL* GetGradOut() {return grad_out;} // return pointer on
output gradient for chaining
  virtual void SetDataIn(REAL *data) {data_in=data;} // set pointer of
input data
  virtual void SetGradOut(REAL *data) {grad_out=data;} // set pointer
of output gradient
  virtual void Info(bool=false, char *txt=(char*)" - ");// display
(detailed) information on machine
    // FILE IO
  static Mach *Read(ifstream&);     // read class from a stream
  void Write(ofstream&); // write content of class to a stream
    // Training
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int =0);
};

#endif
```

B.31 MachLin.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachLin.cpp,v 1.17 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
```

```
#include "MachLin.h"
#include "Blas.h"


MachLin::MachLin(const int p_idim, const int p_odim, const int p_bsize,
const int p_nbfw, const int p_nbbw)
 : Mach(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
  if (odim>0) {
    b = new REAL[odim];
    if (!b) Error ("can't allocate memory for bias of linear machine");
  }
  else b=NULL;
  if (idim*odim>0) {
    w = new REAL[idim*odim];
    if (!w) Error ("can't allocate memory for weights of linear
machine");
  }
  else w=NULL;
}

MachLin::~MachLin()
{

#if 0
  printf("W:\n");
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
  printf("b: ");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
  if (b) delete [] b;
  if (w) delete [] w;
}

void MachLin::BiasConst(const REAL val)
{
  for (int i=0; i<odim; i++) b[i]=val;
}

void MachLin::BiasRandom(const REAL range)
{
  REAL c=range*2.0;
  for (int i=0; i<odim; i++) b[i]=c*(drand48()-0.5);
}

void MachLin::WeightsConst(const REAL val)
{
  for (int i=0; i<idim*odim; i++) w[i]=val;
}

void MachLin::WeightsRandom(const REAL range)
{
  REAL c=range*2.0;
```

```
  for (int i=0; i<idim*odim; i++) w[i]=c*(drand48()-0.5);
}

void MachLin::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on linear machine" << endl;
    Mach::Info(detailed,txt);
  }
  else {
    printf("%sMachLin %d-%d, bs=%d, passes=%d/%d\n", txt, idim, odim,
bsize, nb_forw, nb_backw);
  }
}

//-----------------------------------------------
// File output
//-----------------------------------------------

void MachLin::WriteData(ofstream &outf) {
  int s=odim*idim + odim;
  outf.write((char*) &s,sizeof(int));
  s=sizeof(REAL);
  outf.write((char*) &s,sizeof(int));
  outf.write((char*) w,odim*idim*sizeof(REAL));
  outf.write((char*) b,odim*sizeof(REAL));
#if 0
cout << "\nWriting on file:" << endl;
  printf("W: %dx%d\n",odim,idim);
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
  printf("b:\n");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
}

//-----------------------------------------------
// File input
//-----------------------------------------------


void MachLin::ReadData(ifstream &inpf, size_t s)
{
  size_t se=odim*idim + odim;
  if (s!=se) {
    cerr << "ERROR: data block of linear machine has " << s << "
elements (" << se << " were expected)" << endl;    Error();
  }
  Mach::ReadData(inpf, 0);
    // read parameters
    // TODO: error checks
  inpf.read((char*) w,odim*idim*sizeof(REAL));
  inpf.read((char*) b,odim*sizeof(REAL));
#if 0
```

```
cout << "\nRead from file:" << endl;
  printf("W: %dx%d\n",odim,idim);
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
  printf("b:\n");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
}


//-------------------------------------------------
// Training
//-------------------------------------------------

void MachLin::Forw(int eff_bsize)
{
  if (!data_in)
    Error("MachLin::Forw(): input data is not set");

  if (eff_bsize<=0) eff_bsize=bsize;

#if 0
  printf("Forw %p, bsize=%d\n", (void*)this, eff_bsize);
  printf("W: %dx%d\n",odim,idim);
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
  printf("b:\n");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
#if 0
for (int e=0; e<eff_bsize; e++) {
  printf("B %d inp:", e);
  for (int i=0; i<idim; i++) printf(" %7.5f", data_in[i+e*idim]);
  printf("\n");
}
#endif

#ifdef BLAS
  if (eff_bsize>1) { // BLAS block mode: GEMM
    int e,o;
    REAL *optr, *bptr;

      // copy bias <eff_bsize> times into result matrix
    for (e=0, optr=data_out; e<eff_bsize; e++) {
      for (o=0, bptr=b; o<odim; o++) *optr++ = *bptr++;
    }
    call_gemm (data_out, w, data_in, 1.0, odim, eff_bsize, idim);
  }
  else { // BLAS vector mode: GEMV
    call_gemv (data_out, w, data_in, b, odim, idim);
  }
```

```
#else
  for (int e=0; e<eff_bsize; e++) {
      // simple matrix vector multiply, TODO: verify bsize
      // TODO: W is stored in BLAS (Fortan) format: colum major !!
    //cout << "forw ex " << e << endl;
    REAL *wptr=w;
    for (int o=0; o<odim; o++) {
      REAL s=b[o];
      for (int i=0; i<idim; i++) s+=wptr[i*odim+o]*data_in[i+e*idim];
      data_out[o+e*odim]=s;
    }
  }
#endif
  nb_forw += eff_bsize;

#if 0
for (int e=0; e<eff_bsize; e++) {
  printf("B %d out:", e);
  for (int i=0; i<odim; i++) printf(" %7.5f", data_out[i+e*odim]);
  printf("\n\n");
}
#endif

}


void MachLin::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  static REAL real1=1.0, real0=0.0;
  static char transN='N', transT='T';
  REAL epsilon = 1.0 + lrate * wdecay;

  if (eff_bsize<=0) eff_bsize=bsize;
  if (!grad_out)
    Error("MachLin::Backw(): output gradient is not set");

#if 0
for (int e=0; e<eff_bsize; e++) {
  printf(" B %d grad:", e);
  for (int i=0; i<idim; i++) printf(" %7.5f", grad_out[i]);
  printf("\n");
}
#endif

    // update bias vector:   b = b + lrate * grad_out
    // NO weight decay
  REAL *gptr = grad_out;
  for (int e=0; e<eff_bsize; e++) {
    REAL *aptr = b;
    for (int i=0; i<odim; i++) *aptr++ += lrate * *gptr++;
  }

#if 0
  printf("b after update:\n");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
```

```
#endif

    // backprop gradient:   grad_in   =        w'        *   grad_out
    //                    idim x bsize = (odim x idim)'  *  odim x
bsize
//printf("GEMM(%lx=%lx * % x)\n",grad_in, w, grad_out);
  GEMM (&transT, &transN, &idim, &eff_bsize, &odim,
        &real1, w, &odim, grad_out, &odim,
        &real0, grad_in, &idim);

    // update weights including weight decay
    // w = lrate  *grad_out * data_in^T + epsilon * w
    // gemm (transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c,
ldc )
    //                                       Go      Din            W
    //        C = alpha*A * B + beta * b
    //
#if 0
  printf("W before update:\n");
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
#endif
//printf("GEMM(%lx=%lx * % x)\n",w, grad_out, data_in);
  GEMM (&transN, &transT, &odim, &idim, &eff_bsize,
        &lrate, grad_out, &odim, data_in, &idim,
        &epsilon, w, &odim);
#if 0
  printf("W after update:\n");
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
#endif
  nb_backw += eff_bsize;
}

void MachLin::Debug()
{
  for (int o=0; o<odim; o++) {
    for (int i=0; i<idim; i++) {
      w[i*odim+o] = i + 1000*o;
    }
    b[o] = -o;
  }
}
```

B.32 MachLin.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
```

```
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachLin.h,v 1.12 2010/01/25 12:27:07 schwenk Exp $
 *
 * linear machine:  output = weights * input + biases
 */

#ifndef _MachLin_h
#define _MachLin_h

#include "Mach.h"

class MachLin : public Mach
{
protected:
  REAL *b;        // biases
  REAL *w;        // weights, stored in BLAS format, e.g. COLUMN major
!
  virtual void ReadData(ifstream&, size_t); // read binary data
  virtual void WriteData(ofstream&); // write binary data
public:
  MachLin(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~MachLin();
  virtual int GetMType() {return file_header_mtype_lin;};   // get type
of machine
  virtual void BiasConst(const REAL val); // init biases with constant
values
  virtual void BiasRandom(const REAL range);    // random init of
biases in [-range, range]
  virtual void WeightsConst(const REAL val);    // init weights with
constant values
  virtual void WeightsRandom(const REAL range); // random init of
weights in [-range, range]
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
  virtual void Debug ();
```

```
};

#endif
```

B.33 MachMulti.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachMulti.cpp,v 1.14 2010/01/28 09:27:12 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "MachMulti.h"

MachMulti::MachMulti()
 : Mach(0,0,0)
{
  machs.clear();
}

MachMulti::~MachMulti()
{
  machs.clear();
}

void MachMulti::Delete()
{
  for (unsigned int m=0; m<machs.size(); m++) delete machs[m];
}
```

```
void MachMulti::MachAdd(Mach *new_mach)
{
  Error("MachAdd not defined for abstract multiple machine");
}

Mach *MachMulti::MachDel()
{
  Error("MachDel not defined for abstract multiple machine");
  return NULL;
}

//-----------------------------------------------
// File output
//-----------------------------------------------


void MachMulti::WriteParams(ofstream &of) {
  Mach::WriteParams(of);
  int nbm=machs.size();
  of.write((char*) &nbm, sizeof(int));
}

void MachMulti::WriteData(ofstream &outf) {
  int nbm=machs.size(), s=sizeof(REAL);
  outf.write((char*) &nbm, sizeof(int));
  outf.write((char*) &s, sizeof(int));
  for (vector<Mach*>::iterator it = machs.begin(); it!=machs.end();
++it) {
    (*it)->Write(outf);
  }
}

//-----------------------------------------------
// File input
//-----------------------------------------------

void MachMulti::ReadParams(ifstream &inpf, bool with_alloc)
{
  if (machs.size() > 0)
    Error("Trying to read multiple machine into non empty data
structures\n");

  Mach::ReadParams(inpf, false);
  int nbm;
  inpf.read((char*) &nbm, sizeof(int));
  if (nbm<1) Error("illegal number of machines");
  machs.clear();
  for (int i=0; i<nbm; i++) machs.push_back(NULL);
}

void MachMulti::ReadData(ifstream &inpf, size_t s)
{
  if (s!=machs.size()) {
    cerr << "ERROR: data block of multiple machine has " << s << "
machines (" << machs.size() << " were expected)" << endl;    Error();
  }
```

```
  for (vector<Mach*>::iterator it = machs.begin(); it!=machs.end();
++it) {
    (*it) = Mach::Read(inpf);
  }
}

//
// Tools
//

void MachMulti::SetBsize(int bs)
{
  if (bs<1) Error("wrong value in SetBsize()");
  for (uint i=0; i<machs.size(); i++) machs[i]->SetBsize(bs);
}

void MachMulti::Info(bool detailed, char *txt)
{
  if (detailed) {
    if (machs.size()) {
      Mach::Info();
      for (unsigned int i=0; i<machs.size(); i++) {
        cout << "MACHINE " << i << ": " << endl;
        machs[i]->Info();
      }
    }
    else
      cout << " *** empty ***" << endl;
  }
  else {
    printf("%sMultiple machine %d- .. -%d, bs=%d, passes=%d/%d\n", txt,
idim, odim, bsize, nb_forw, nb_backw);
    char ntxt[256];
    sprintf(ntxt,"%s  ", txt);
    for (unsigned int i=0; i<machs.size(); i++) machs[i]-
>Info(detailed, ntxt);
  }
}

void MachMulti::Forw(int eff_bszie)
{
  if (machs.empty())
    Error("called Forw() for an empty multiple machine");
  else
    Error("call to Forw() not defined for an abstract multiple
machine");
}

void MachMulti::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  Error("call to Backw() not defined for an abstract multiple
machine");
}
```

B.34 MachMulti.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachMulti.h,v 1.10 2010/01/26 19:37:22 schwenk Exp $
 *
 * virtual class to support various combinations of multiple machines
 */

#ifndef _MachMulti_h
#define _MachMulti_h

using namespace std;
#include <vector>

#include "Mach.h"

class MachMulti : public Mach
{
protected:
  vector<Mach*> machs;
  virtual void ReadParams(ifstream&, bool =true);
  virtual void ReadData(ifstream&, size_t); // read binary data
  virtual void WriteParams(ofstream&); // write all params
  virtual void WriteData(ofstream&); // write binary data
public:
  MachMulti();    // create initial sequence with no machine
  virtual ~MachMulti();
  virtual int GetMType() {return file_header_mtype_multi;}; // get type
of machine
  void SetBsize(int bs);
    // add and remove machines
  virtual void Delete();      // call destructor for all the machines
```

```
  virtual void MachAdd(Mach*);      // add new machine after the
existing ones
  virtual Mach *MachDel();    // delete the last machine
    // standard functions
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);  // calculate outputs for current inputs
  virtual void Backw(const float lrate, const float wdecay, int=0); //
calculate gradients at input for current gradients at output
};

#endif
```

B.35 MachPar.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachPar.cpp,v 1.12 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "MachTab.h"
#include "MachPar.h"

void MachPar::do_alloc()
{
  if (data_out) delete [] data_out;
  if (grad_in) delete [] grad_in;
  data_out = (odim*bsize>0) ? new REAL[odim*bsize] : NULL;
  grad_in = (idim*bsize>0) ? new REAL[idim*bsize] : NULL;
```

```
}


MachPar::MachPar()
 : MachMulti()
{
}

MachPar::~MachPar()
{
  // data_out and grad_in will be freed by Mach::~Mach()
}

void MachPar::MachAdd(Mach *new_mach)
{
  if (machs.empty()) {
    machs.push_back(new_mach);
      // think about freeing memory
    idim=new_mach->GetIdim();
    odim=new_mach->GetOdim();
    bsize=new_mach->GetBsize();
    data_in=NULL; // will be set by MachPar::SetDataIn()
    data_out=NULL;
    grad_in = NULL;
    grad_out = NULL;
    do_alloc();
    new_mach->SetGradOut(grad_out);
  }
  else {
    if (bsize!=new_mach->GetBsize())
      Error("bunch size of new parallel machine does not match");
    machs.push_back(new_mach);

      // resize input gradient and output data
    idim += new_mach->GetIdim();
    odim += new_mach->GetOdim();
    do_alloc();
  }
}

Mach *MachPar::MachDel()
{
  if (machs.empty()) {
    Error("impossible to delete element from parallel machine: is
already empty");
  }

  Error("TODO");
  return NULL;
}

// set pointer of input data
void MachPar::SetDataIn(REAL *data)
{
  data_in=data;
    // set input data of indiv machines one after each other
    // this depends on the effective bsize !
```

```
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->SetDataIn(data);
    data += bsize*machs[m]->GetIdim();
  }
}

// set pointer of output gradient
void MachPar::SetGradOut(REAL *data)
{
  grad_out=data;
    // set output gradients of indiv machines one after each other
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->SetGradOut(data);
    data += bsize*machs[m]->GetOdim();
  }
}


//-----------------------------------------------
// File output
//-----------------------------------------------

void MachPar::ReadData(ifstream &inpf, size_t s)
{
  MachMulti::ReadData(inpf,s);

     // calculate idim and odim and and allocate data_out and grad_in
  idim=odim=0;
  for (uint m=0; m<machs.size(); m++) {
    idim += machs[m]->GetIdim();
    odim += machs[m]->GetOdim();
  }
  bsize = machs[0]->GetBsize();
  do_alloc();

    // scanning for MachTab with shared addresses
  REAL *tadr=NULL;
  for (uint m=0; m<machs.size(); m++) {
    MachTab *mt= (MachTab*) machs[m];
    if (mt->GetMType()==file_header_mtype_tab) {
      if (mt->GetTabAdr()) {
        if (tadr) {
        }
        else {
        }
        tadr=mt->GetTabAdr();
      }
      else {
        mt->SetTabAdr(tadr);
      }
    }
  }
}

//
// Tools
//
```

```
void MachPar::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on parallel machine" << endl;
    MachMulti::Info(detailed);
  }
  else {
    printf("%sParallel machine %d- .. %d, bs=%d, passes=%d/%d\n", txt,
idim, odim, bsize, nb_forw, nb_backw);
    char ntxt[256];
    sprintf(ntxt,"%s  ", txt);
    for (unsigned int i=0; i<machs.size(); i++) machs[i]-
>Info(detailed, ntxt);
  }
}


// forward pass for all machines and copy output into cumulated output
void MachPar::Forw(int eff_bsize)
{
  if (machs.empty())
    Error("called Forw() for an empty parallel machine");

  if (eff_bsize<=0) eff_bsize=bsize;

      // we need to set the pointers to the input data of indiv
machines
      // one after each other since this depends on the effective bsize
!

  REAL *iptr=data_in;
  REAL *optr=data_out;
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->SetDataIn(iptr);
    machs[m]->Forw(eff_bsize);
    memcpy(optr, machs[m]->GetDataOut(), eff_bsize*machs[m]-
>GetOdim()*sizeof(REAL));
    iptr += eff_bsize*machs[m]->GetIdim();
    optr += eff_bsize*machs[m]->GetOdim();
  }
  nb_forw += eff_bsize;
}

// backward pass for all machines and copy input gradient into
cumulated gradient
void MachPar::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  if (machs.empty())
    Error("called Backw() for an empty parallel machine");
  if (eff_bsize<=0) eff_bsize=bsize;

      // we need to set the pointers to output gradients of indiv
machines
      // one after each other since this depends on the effective bsize
!
```

```
  REAL *gptr=grad_in;
  REAL *optr=grad_out;
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->SetGradOut(optr);
    machs[m]->Backw(lrate,wdecay,eff_bsize);
    memcpy(gptr, machs[m]->GetGradIn(), eff_bsize*machs[m]-
>GetIdim()*sizeof(REAL));
    optr += eff_bsize*machs[m]->GetOdim();
    gptr += eff_bsize*machs[m]->GetIdim();
  }
  nb_backw += eff_bsize;
}
```

B.36 MachPar.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachPar.h,v 1.9 2010/01/25 12:27:07 schwenk Exp $
 *
 *  Parallel machine:
 *   - put several machine in parallel with a concatenated input and
output layer
 *   - the dimensions of the input and output layers may be different
 */

#ifndef _MachPar_h
#define _MachPar_h

using namespace std;
#include <vector>

#include "MachMulti.h"
```

```
class MachPar : public MachMulti
{
private:
  void do_alloc();      // perform allocation of dynamic data
structures
protected:
  virtual void ReadData(ifstream&, size_t); // read binary data
public:
  MachPar();      // create initial sequence with no machine
  virtual ~MachPar();
  virtual int GetMType() {return file_header_mtype_mpar;};  // get type
of machine
    // redfine connecting functions
  virtual void SetDataIn(REAL*);    // set pointer of input data
  virtual void SetGradOut(REAL*);   // set pointer of output gradient
    // add and remove machines
  virtual void MachAdd(Mach*); // add new machine after the existing
ones
  virtual Mach *MachDel();
    // standard functions
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
  virtual void Backw(const float lrate, const float wdecay, int=0);
      // calculate gradients at input for current gradients at output
};

#endif
```

B.37 MachSeq.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

```
 *
 * $Id: MachSeq.cpp,v 1.14 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "MachSeq.h"

MachSeq::MachSeq()
 : MachMulti()
{
}

MachSeq::~MachSeq()
{
  data_out=grad_in=NULL;  // prevent delete[] by ~Mach()
}

// set pointer of input data
void MachSeq::SetDataIn(REAL *data)
{
  data_in=data;
  if (machs.size() > 0) machs[0]->SetDataIn(data);
}

// set pointer of output gradient
void MachSeq::SetGradOut(REAL *data)
{
  grad_out=data;
  if (machs.size() > 0) machs.back()->SetGradOut(data);
}

void MachSeq::MachAdd(Mach *new_mach)
{
  if (machs.empty()) {
    machs.push_back(new_mach);
      // think about freeing memory
    idim=new_mach->GetIdim();
    bsize=new_mach->GetBsize();
    data_in=new_mach->GetDataIn();
    grad_in=new_mach->GetGradIn();
  }
  else {
    Mach *last_mach=machs.back();
    if (last_mach->GetOdim()!=new_mach->GetIdim()) {
      cout << "Current sequential machine:" << endl; Info(false);
      cout << "Newly added machine:" << endl; new_mach->Info(false);
      Error("input dimension of new sequential machine does not
match");
    }
    if (bsize!=new_mach->GetBsize()) {
      cout << "Current sequential machine:" << endl; Info(false);
      cout << "Newly added machine:" << endl; new_mach->Info(false);
      Error("bunch size of new sequential machine does not match");
    }
```

```
    machs.push_back(new_mach);

      // connect new last machine to the previous one
    new_mach->SetDataIn(last_mach->GetDataOut());
    last_mach->SetGradOut(new_mach->GetGradIn());
  }

    // connect last machine to the outside world
  odim=new_mach->GetOdim();
  data_out=new_mach->GetDataOut();
  grad_out=new_mach->GetGradOut();
}

Mach *MachSeq::MachDel()
{
  if (machs.empty()) {
    Error("impossible to delete element from sequential machine: is
already empty");
  }

  Mach *del_mach=machs.back();
  machs.pop_back();

  if (machs.empty()) {
    idim=odim=bsize=0;
    data_in=data_out=grad_in=grad_out=NULL;
  }
  else {
    Mach *last_mach=machs.back();

      // connect new last machine to the outside world
    odim=last_mach->GetOdim();
    data_out=last_mach->GetDataOut();
    grad_out=last_mach->GetGradOut();
  }

  return del_mach;
}

//-----------------------------------------------
// File input
//-----------------------------------------------

void MachSeq::ReadData(ifstream &inpf, size_t s)
{
  MachMulti::ReadData(inpf,s);


  int nbm=machs.size();
  idim = machs[0]->GetIdim();
  bsize = machs[0]->GetBsize();
  odim = machs[nbm-1]->GetOdim();

    // connect first to the outside world
  data_in=machs[0]->GetDataIn();
  grad_in=machs[0]->GetGradIn();
```

```
    // forward chain the data
  for (int m=1; m<nbm; m++) machs[m]->SetDataIn(machs[m-1]-
>GetDataOut());
    // backward chain the gradients
  for (int m=nbm-1; m>0; m--) machs[m-1]->SetGradOut(machs[m]-
>GetGradIn());

    // connect last machine to the outside world
  data_out=machs[nbm-1]->GetDataOut();
  grad_out=machs[nbm-1]->GetGradOut();
}

//
// Tools
//

void MachSeq::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on stacked machine" << endl;
    MachMulti::Info(detailed,txt);
  }
  else {
    printf("%sSequential machine [%u] %d- .. -%d, bs=%d,
passes=%d/%d\n", txt, (uint) machs.size(), idim, odim, bsize, nb_forw,
nb_backw);
    char ntxt[256];
    sprintf(ntxt,"%s  ", txt);
    for (unsigned int i=0; i<machs.size(); i++) machs[i]-
>Info(detailed, ntxt);
  }
}

void MachSeq::Forw(int eff_bsize)
{
  if (machs.empty())
    Error("called Forw() for an empty sequantial machine");
  for (unsigned int i=0; i<machs.size(); i++) machs[i]-
>Forw(eff_bsize);
  nb_forw += (eff_bsize<=0) ? bsize : eff_bsize;
}

void MachSeq::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  if (machs.empty())
    Error("called Backw() for an empty sequantial machine");
  for (int i=machs.size()-1; i>=0; i--) {
      machs[i]->Backw(lrate,wdecay,eff_bsize);
  }
  nb_backw += (eff_bsize<=0) ? bsize : eff_bsize;
}
```

B.38 MachSeq.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSeq.h,v 1.8 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _MachSeq_h
#define _MachSeq_h

using namespace std;
#include <vector>

#include "MachMulti.h"

class MachSeq : public MachMulti
{
protected:
  virtual void ReadData(ifstream&, size_t); // read binary data
public:
  MachSeq();      // create initial sequence with no machine
  virtual ~MachSeq();
  virtual int GetMType() {return file_header_mtype_mseq;};  // get type
of machine
    // redfine connecting functions
  virtual void SetDataIn(REAL*);    // set pointer of input data
  virtual void SetGradOut(REAL*);   // set pointer of output gradient
    // add and remove machines
  virtual void MachAdd(Mach*); // add new machine after the existing
ones
  virtual Mach *MachDel();
    // standard functions
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
```

```
  virtual void Forw(int=0);   // calculate outputs for current inputs
  virtual void Backw(const float lrate, const float wdecay, int=0);
      // calculate gradients at input for current gradients at output
};

#endif
```

B.39 MachSig.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSig.cpp,v 1.10 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <math.h>
//extern double drand48();

#include "Tools.h"
#include "MachSig.h"

MachSig::MachSig(const int p_idim, const int p_odim, const int p_bsize,
const int p_nbfw, const int p_nbbw)
 : MachLin(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
}

MachSig::~MachSig()
{
  printf("** destructor MachSig %lx\n",(luint) this);
}
```

```
//---------------------------------------------
// Tools
//---------------------------------------------

void MachSig::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on sigmoidal machine" << endl;
    MachLin::Info(detailed,txt);
  }
  else {
    printf("%sMachSig %d-%d, bs=%d, passes=%d/%d\n", txt, idim, odim,
bsize, nb_forw, nb_backw);
  }
}

//---------------------------------------------
// Training
//---------------------------------------------

void MachSig::Forw(int eff_bsize)
{
  if (eff_bsize<=0) eff_bsize=bsize;
  MachLin::Forw(eff_bsize);

    // apply sigmoid on output
#ifdef BLAS
  Error("implement sigmoid\n");
#else
  Error("implement sigmoid\n");
#endif
}

void MachSig::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
    // derivate sigmoidal activation function
    //             = grad_hidden .* ( 1 - a_hidden^2 )

  REAL *aptr = data_out;
  REAL *gptr = grad_out;

  if (eff_bsize<=0) eff_bsize=bsize;
  if (!grad_out)
    Error("MachSig::Backw(): output gradient is not set");

  for (int i=0; i<odim*eff_bsize; i++) {
    REAL val = *aptr++;
    Error("implement derivative of sigmoid\n");
    *gptr=val;
  }

  MachLin::Backw(lrate, wdecay, eff_bsize);
}
```

B.40 MachSig.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSig.h,v 1.7 2010/01/25 12:27:07 schwenk Exp $
 *
 * sigmoidal machine:  output = tanh(weights * input + biases)
 */

#ifndef _MachSig_h
#define _MachSig_h

#include "MachLin.h"

class MachSig : public MachLin
{
public:
  MachSig(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~MachSig();
  virtual int GetMType() {return file_header_mtype_sig;};   // get type
of machine
  virtual void Info(bool=false, char *txt= (char*)"");      // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
};

#endif
```

B.41 MachSoftmax.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSoftmax.cpp,v 1.16 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <math.h>
//extern double drand48();

#include "Tools.h"
#include "MachSoftmax.h"
#include "Blas.h"

MachSoftmax::MachSoftmax(const int p_idim, const int p_odim, const int
p_bsize, const int p_nbfw, const int p_nbbw)
 : MachLin(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
}

MachSoftmax::~MachSoftmax()
{
}

//-----------------------------------------------
// Tools
//-----------------------------------------------

void MachSoftmax::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on softmax machine" << endl;
```

```
    MachLin::Info(detailed);
  }
  else {
    printf("%sMachSoftmax %d-%d, bs=%d, passes=%d/%d\n", txt,idim,
odim, bsize, nb_forw, nb_backw);
  }
}

//------------------------------------------------
// Training
//------------------------------------------------

void MachSoftmax::Forw(int eff_bsize)
{
  if (eff_bsize<=0) eff_bsize=bsize;
  MachLin::Forw(eff_bsize);

    // apply exp() on output and normalize
#ifdef BLAS_INTEL_MKL
  int s=eff_bsize*odim;
  VEXP(&s,data_out,data_out);
  REAL *optr=data_out;
  for (int b=0; b<eff_bsize; b++) {
    REAL sum=0; // TODO: double
    for (int i=0; i<odim; i++) sum += *optr++;
    optr-=odim;
    sum = 1.0/sum;  // circumvent division in loop
    for (int i=0; i<odim; i++) *optr++ *= sum;
  }
#else
  REAL *optr=data_out;
  for (int b=0; b<eff_bsize; b++) {
    REAL sum=0; // TODO: double
    for (int i=0; i<odim; i++) {
      *optr = exp(*optr);
      sum += *optr++;
    }
    optr-=odim;
    sum = 1.0/sum;  // circumvent division in loop
    for (int i=0; i<odim; i++) *optr++ *= sum;
  }
#endif
}

void MachSoftmax::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
    // derivate softmax activation function
    //   do_i / da_k = o_i (kronecker_ik - o_k)
    // we suppose that do_i/da_k vanishes in the error function !!
    //             = o_i (1 - o_i)

#if 0
   // this can't be done here since the result depends
   // on the error function (we must derivate each output w/r
   // to ALL other outputs. This can't be stored in one vector)
   //   dE/da_i = sum_k dE/do_k do_k/da_i
```

```
   // On the other hand, many terms vanish with usual error functions

  REAL *aptr = data_out;
  REAL *gptr = grad_out;

  if (eff_bsize<=0) eff_bsize=bsize;
  if (!grad_out)
    Error("MachSoftmax::Backw(): output gradient is not set");

  for (int b=0; b<eff_bsize; b++) {
    REAL o;
    for (int i=0; i<odim; i++) {
      o=*optr++;
      *gptr++ *= o * (1-o);
    }
  }
#endif

  MachLin::Backw(lrate, wdecay, eff_bsize);
}
```

B.42 MachSoftmax.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSoftmax.h,v 1.8 2010/01/25 12:27:07 schwenk Exp $
 *
 * softmax machine:  a_i = exp(a_i) / sum_k a_k
 * with a_k is the kth output of a linear machine
 */

#ifndef _MachSoftmax_h
#define _MachSoftmax_h
```

```
#include "MachLin.h"

class MachSoftmax : public MachLin
{
public:
  MachSoftmax(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~MachSoftmax();
  virtual int GetMType() {return file_header_mtype_softmax;};     //
get type of machine
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
};

#endif
```

B.43 MachStacked.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachStacked.cpp,v 1.7 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "MachStacked.h"
```

```
MachStacked::MachStacked()
 : MachMulti()
{
}

MachStacked::~MachStacked()
{
  // data_out and grad_in will be freed by Mach::~Mach()
}

void MachStacked::MachAdd(Mach *new_mach)
{
  if (machs.empty()) {
    machs.push_back(new_mach);
      // think about freeing memory
    idim=new_mach->GetIdim();
    odim=new_mach->GetOdim();
    bsize=new_mach->GetBsize();
    data_in=NULL; // will be set by MachStacked::SetDataIn()
    data_out = (odim*bsize>0) ? new REAL[odim*bsize] : NULL;
    grad_in = (idim*bsize>0) ? new REAL[idim*bsize] : NULL;
    grad_out = NULL;
    new_mach->SetGradOut(grad_out);
  }
  else {
    if (bsize!=new_mach->GetBsize())
      Error("bunch size of new stacked machine does not match");
    if (idim!=new_mach->GetIdim())
      Error("input dimension of new stacked machine does not match");
    machs.push_back(new_mach);

      // resize output
    odim += new_mach->GetOdim();
    if (data_out) delete [] data_out;
    data_out = (odim*bsize>0) ? new REAL[odim*bsize] : NULL;
    new_mach->SetDataIn(data_in);
  }
}

Mach *MachStacked::MachDel()
{
  if (machs.empty()) {
    Error("impossible to delete element from stacked machine: is
already empty");
  }

  Mach *del_mach=machs.back();
  machs.pop_back();

  if (machs.empty()) {
    idim=odim=bsize=0;
    if (data_out) delete [] data_out;
    if (grad_in) delete [] grad_in;
    data_in=data_out=grad_in=grad_out=NULL;
  }
  else {
    Mach *last_mach=machs.back();
```

```
      // connect new last machine to the outside world
    odim=last_mach->GetOdim();
    data_out=last_mach->GetDataOut();
    grad_out=last_mach->GetGradOut();
  }

  return del_mach;
}

// set pointer of input data
void MachStacked::SetDataIn(REAL *data)
{
  data_in=data;
    // all machines point on the same input
  for (unsigned int m=0; m<machs.size(); m++) machs[m]-
>SetDataIn(data_in);
}

// set pointer of output gradient
void MachStacked::SetGradOut(REAL *data)
{
  grad_out=data;

    // set output gradients of inidv machines one after each other
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->SetGradOut(data);
    data += machs[m]->GetOdim();
  }
}


//-----------------------------------------------
// File output
//-----------------------------------------------

void MachStacked::WriteParams(ofstream &of) {
  Mach::WriteParams(of);
  of << file_header_name_nbmach << " " << machs.size() << endl;
}

void MachStacked::WriteData(ofstream &outf) {
  outf << file_header_name_databeg << " " << machs.size() << endl;
  for (vector<Mach*>::iterator it = machs.begin(); it!=machs.end();
++it) {
    (*it)->Write(outf);
  }
}

//-----------------------------------------------
// File input
//-----------------------------------------------

void MachStacked::ReadParams(ifstream &inpf)
{
  int nbm=ReadInt(inpf, file_header_name_nbmach, 1);
}
```

```
void MachStacked::ReadData(ifstream &inpf, size_t s)
{
  if (s!=machs.size()) {
    cerr << "ERROR: data block of multiple machine has " << s << "
machines (" << machs.size() << " were expected)" << endl;    Error();
  }

  for (vector<Mach*>::iterator it = machs.begin(); it!=machs.end();
++it) {
    (*it)->Read(inpf);
  }
}

//
// Tools
//

void MachStacked::Info(bool detailed)
{
  if (detailed) {
    cout << "Information on stacked machine" << endl;
    MachMulti::Info(detailed);
  }
  else {
    printf(" - Stacked machine %d- .. %d, bs=%d, passes=%d/%d\n", idim,
odim, bsize, nb_forw, nb_backw);
    for (unsigned int i=0; i<machs.size(); i++) machs[i]-
>Info(detailed);
  }
}


// forward pass for all machines and copy output into cumulated output
void MachStacked::Forw(int eff_bsize)
{
  if (machs.empty())
    Error("called Forw() for an empty stacked machine");
  REAL *optr=data_out;
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->Forw(eff_bsize);
    memcpy(optr, machs[m]->GetDataOut(), machs[m]-
>GetOdim()*sizeof(REAL));
    optr+=machs[m]->GetOdim();
  }
  nb_forw += (eff_bsize<=0) ? bsize : eff_bsize;
}

// backward pass for all machines and cumulate gradient at input
void MachStacked::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  if (machs.empty())
    Error("called Backw() for an empty sequantial machine");
  machs[0]->Backw(lrate,wdecay,eff_bsize);
  memcpy(grad_in, machs[0]->GetGradIn(), idim*sizeof(REAL));
  for (unsigned int m=1; m<machs.size(); m++) {
```

```
    machs[m]->Backw(lrate,wdecay,eff_bsize);
    for (int i=0; i<idim; i++) grad_in[i] += machs[m]->GetGradIn()[i];
  }
  nb_backw += (eff_bsize<=0) ? bsize : eff_bsize;
}
```

B.44 MachStacked.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachStacked.h,v 1.8 2010/01/25 12:27:07 schwenk Exp $
 *
 *  All the indiv machines must have the same input dimension and
bsize, but
 *  the output dimension can vary
 *  This machine allocates memory for the output data which cumulates
the
 *  values from the machines
 *  All indiv machines share the same pointer on the input (not
allocated by the
 *  stacked machine). We allocate an input gradient which cumulates the
gradients
 *  of the individual machines
 */

#ifndef _MachStacked_h
#define _MachStacked_h

using namespace std;
#include <vector>

#include "MachMulti.h"
```

```
class MachStacked : public MachMulti
{
protected:
  virtual void ReadParams(ifstream&); // read all params
  virtual void ReadData(ifstream&, size_t); // read binary data
  virtual void WriteParams(ofstream&); // write all params
  virtual void WriteData(ofstream&); // write binary data
public:
  MachStacked();  // create initial sequence with no machine
  virtual ~MachStacked();
  virtual int GetMType() {return file_header_mtype_mstack;};      //
get type of machine
    // redfine connecting functions
  virtual void SetDataIn(REAL*);    // set pointer of input data
  virtual void SetGradOut(REAL*);   // set pointer of output gradient
    // add and remove machines
  virtual void MachAdd(Mach*); // add new machine after the existing
ones
  virtual Mach *MachDel();
    // standard functions
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
  virtual void Backw(const float lrate, const float wdecay, int=0);
      // calculate gradients at input for current gradients at output
};

#endif
```

B.45 MachTab.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
```

```
 * $Id: MachTab.cpp,v 1.14 2010/01/26 19:37:22 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
#include "MachTab.h"

void MachTab::do_alloc()
{
  if (!ext_alloc) {
    t = new REAL[idim*odim];
    if (!t) Error ("can't allocate memory for table look-up machine");
  }
  else
    ;
}


MachTab::MachTab(const int p_idim, const int p_odim, const int p_bsize,
const int p_nbfw, const int p_nbbw)
 : Mach(1, p_odim, p_bsize, p_nbfw, p_nbbw), ext_alloc(false)
{
  if (p_idim<=0) Error("Table machine: illegal value of input
dimension");
  if (p_odim<=0) Error("Table machine: illegal value of output
dimension");
  idim = p_idim; // override 1 in call to Mach()

  do_alloc();
}

MachTab::MachTab(REAL *ext_table,
      const int p_idim, const int p_odim, const int p_bsize,
      const int p_nbfw, const int p_nbbw)
 : Mach(1, p_odim, p_bsize, p_nbfw, p_nbbw), ext_alloc(true)
{
  if (p_idim<0) Error("Table machine: illegal value of input
dimension");
  if (p_odim<0) Error("Table machine: illegal value of output
dimension");
  idim = p_idim; // override 1 in call to Mach()

  //if (!ext_table) Error ("Table look-up machine: provided address is
NULL");
  t=ext_table;
  do_alloc();
}

MachTab::~MachTab()
{

  if (!ext_alloc & (t!=NULL)) delete [] t;
}
```

```
void MachTab::TableConst(const REAL val)
{
  for (int i=0; i<idim*odim; i++) t[i]=val;
}

void MachTab::TableRandom(const REAL range)
{
  REAL c=range*2.0;
  for (int i=0; i<idim*odim; i++) t[i]=c*(drand48()-0.5);
}

void MachTab::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on table look-up machine" << endl;
    Mach::Info(detailed,txt);
  }
  else {
    printf("%sMachTab 1[%d]-%d, bs=%d, passes=%d/%d\n", txt, idim,
odim, bsize, nb_forw, nb_backw);
  }
}

//-----------------------------------------------
// File output
//-----------------------------------------------

void MachTab::WriteParams(ofstream &of)
{

  Mach::WriteParams(of);
  of.write((char*) &ext_alloc, sizeof(int));
}


void MachTab::WriteData(ofstream &outf) {
  int i=0, s=sizeof(REAL);
  if (ext_alloc) {
    outf.write((char*) &i, sizeof(int));
    outf.write((char*) &s, sizeof(int));
  }
  else {
    i=idim*odim;
    outf.write((char*) &i, sizeof(int));
    outf.write((char*) &s, sizeof(int));
    outf.write((char*) t,odim*idim*sizeof(REAL));
  }
}

//-----------------------------------------------
// File input
//-----------------------------------------------

void MachTab::ReadParams(ifstream &inpf, bool with_alloc)
{
```

```
  Mach::ReadParams(inpf, false);
  inpf.read((char*) &ext_alloc, sizeof(int));
  do_alloc();
}

void MachTab::ReadData(ifstream &inpf, size_t s)
{
  size_t se=odim*idim;

  if (ext_alloc) {
    if (s>0) {
      fprintf(stderr,"internal error in file, table look-up machine has
external allocation, but %u elements of data are provided\n",(uint)s);
      Error();
    }
    return; // address will be filled in by MachPar
  }
  else if (s!=se) {
    fprintf(stderr,"data block of table look-up machine has %u elements
- %u were expected)",(uint) s, (uint) se);
    Error();
  }
  Mach::ReadData(inpf, 0);
  inpf.read((char*) t,odim*idim*sizeof(REAL));
}


//-----------------------------------------------
// Training
//-----------------------------------------------

void MachTab::Forw(int eff_bsize)
{
  if (!data_in)
    Error("MachTab::Forw(): input data is not set");

  if (eff_bsize<=0) eff_bsize=bsize;

  REAL *optr=data_out;
  for (int b=0; b<eff_bsize; b++) {
    int idx= (int) data_in[b];
    if (idx<0 || idx>=idim) {
      fprintf(stderr,"ERROR: illegal index (%d) in table look-up
machine, should be in [0,%d[\n", idx, idim);
      Error();
    }
    memcpy(optr, t+idx*odim, odim*sizeof(REAL));
    optr+=odim;
  }
  nb_forw+=eff_bsize;
}


void MachTab::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  // table[wid] = table[wid] + lrate * grad_out[wid] * data_in[wid]
```

```
  REAL *gptr = grad_out;
  for (int b=0; b<eff_bsize; b++) {
    int idx= (int) data_in[b];
    if (idx<0 || idx>=idim) {
      fprintf(stderr,"ERROR: illegal index (%d) in table look-up
machine (backw), should be in [0,%d[\n", idx, idim);
      Error();
    }
    REAL *tptr=t+idx*odim;
#ifdef DEBUG
    printf("  B %d idx=%d\n",b,idx);
    printf("  grad:"); for (int i=idx-2;i<=idx+2;i++) printf("
%f",gptr[i]); printf("\n");
    printf("   tab:");  for (int i=-2;i<=2;i++) printf(" %f",tptr[i]);
printf("\n");
#endif
    for (int i=0; i<odim; i++) *tptr++ += lrate * *gptr++;
    grad_in[b]=0; // we don't backprop to the input of a tbale look-up
machine
  }
}
```

B.46 MachTab.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTab.h,v 1.9 2010/01/25 12:27:07 schwenk Exp $
 *
 * Table lookup machine:
 *   - input = index in table
 *   - output = ith line of table
 */
```

```
#ifndef _MachTab_h
#define _MachTab_h

#include "Mach.h"

class MachTab : public Mach
{
private:
  bool ext_alloc; // flag to indicate whether table was allocated
intertnally
  virtual void do_alloc();    // perform allocation of dynamic data
structures
protected:
  REAL *t;        // look-up table
  virtual void WriteParams(ofstream&);
  virtual void ReadParams(ifstream&, bool =true);
  virtual void ReadData(ifstream&, size_t); // read binary data
  virtual void WriteData(ofstream&); // write binary data
  virtual int GetIdim() {return 1; } // we use idim internally as the
dim of the table entries
public:
  MachTab(const int=1, const int=1, const int=1, const int=0, const
int=0); // TODO: idim,odim init ??
  MachTab(REAL*, const int, const int, const int=1, const int=0, const
int=0);
  virtual ~MachTab();
  virtual int GetMType() {return file_header_mtype_tab;};   // get type
of machine
  virtual void TableConst(const REAL val);      // init table with
constant values
  virtual void TableRandom(const REAL range);   // random init of table
in [-range, range]
  virtual REAL *GetTabAdr() {return t; }  //
  virtual void SetTabAdr(REAL *p_adr) {t=p_adr; }     //
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
};

#endif
```

B.47 MachTabShared.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
```

```
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTabShared.cpp,v 1.8 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
#include "MachTabShared.h"

MachTabShared::MachTabShared(REAL* table,
      const int p_idim, const int p_odim, const int p_bsize,
      const int p_nbfw, const int p_nbbw)
 : Mach(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
  printf("** constructor MachTabShared %lx\n", (luint) this);
  if (p_idim<=0) Error("Table machine: illegal value of input
dimension");
  if (p_odim<=0) Error("Table machine: illegal value of output
dimension");

  if (!table) Error ("Shared table look-up machine: found NULL
pointer");
  t=table;
}

MachTabShared::~MachTabShared()
{
  printf("** destructor MachTabShared %lx\n", (luint) this);

  // do NOT free t[] since it was allocated externally
}

void MachTabShared::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on shared table look-up machine" << endl;
    Mach::Info(detailed);
  }
  else {
```

```
    printf("%sMachTabShared %d-%d, bs=%d, passes=%d/%d\n", txt, idim,
odim, bsize, nb_forw, nb_backw);
  }
}

//-----------------------------------------------
// File output
//-----------------------------------------------

void MachTabShared::WriteData(ofstream &outf) {
  outf << file_header_name_databeg << " 0 " << endl;
  outf << file_header_name_dataend << endl;
}

//-----------------------------------------------
// File input
//-----------------------------------------------

void MachTabShared::ReadData(ifstream &inpf, size_t s)
{
  size_t se=odim*(idim+1)*sizeof(REAL);
  if (s!=0) {
    cerr << "ERROR: data block of shared table look-up machine has " <<
s << " bytes (0 were expected)" << endl; Error();
  }
  Mach::ReadData(inpf, 0);
}
```

B.48 MachTabShared.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTabShared.h,v 1.7 2010/01/25 12:27:07 schwenk Exp $
```

```
 *
 * Table lookup machine:
 *   - input = index in table
 *   - output = ith line of table
 */

#ifndef _MachTabShared_h
#define _MachTabShared_h

#include "Mach.h"

class MachTabShared : public MachTab
{
protected:
  REAL *t;        // look-up table
  virtual void ReadData(ifstream&, size_t); // read binary data
  virtual void WriteData(ofstream&); // write binary data
  virtual int GetIdim() {return 1; } // we use idim as the dim of the
table entries
public:
  MachTabShared(REAL*, const int, const int, const int=1, const int=0,
const int=0);
  virtual ~MachTabShared();
  virtual int GetMType() {return file_header_mtype_stab;};  // get type
of machine
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
};

#endif
```

B.49 MachTanh.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
```

```
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTanh.cpp,v 1.14 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <math.h>
//extern double drand48();

#include "Tools.h"
#include "MachTanh.h"
#include "Blas.h"

MachTanh::MachTanh(const int p_idim, const int p_odim, const int
p_bsize, const int p_nbfw, const int p_nbbw)
 : MachLin(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
}

MachTanh::~MachTanh()
{
}


//------------------------------------------------
// Tools
//------------------------------------------------

void MachTanh::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on tanh machine" << endl;
    MachLin::Info(detailed,txt);
  }
  else {
    printf("%sMachTanh %d-%d, bs=%d, passes=%d/%d\n", txt, idim, odim,
bsize, nb_forw, nb_backw);
  }
}

//------------------------------------------------
// Training
//------------------------------------------------

void MachTanh::Forw(int eff_bsize)
{
  if (eff_bsize<=0) eff_bsize=bsize;
  MachLin::Forw(eff_bsize);

    // apply tanh() on output
#ifdef BLAS_INTEL_MKL
  int s=eff_bsize*odim;
  VTANH(&s,data_out,data_out);
#else
  for (int i=0; i<eff_bsize*odim; i++) data_out[i]=tanh(data_out[i]);
#endif
```

```
}

void MachTanh::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
    // derivate tanh activation function
    // multiply grad_hidden by derivatives of hidden layer activities
(tanh)
    // grad_out = grad_out .* f'(data_out)
    //          = grad_out .* ( 1 - data_out^2 )

  REAL *aptr = data_out;
  REAL *gptr = grad_out;

  if (eff_bsize<=0) eff_bsize=bsize;
  if (!grad_out)
    Error("MachTanh::Backw(): output gradient is not set");
  for (int i=0; i<odim*eff_bsize; i++) {
    REAL val = *aptr++;
    *gptr++ *= (1.0 - val * val);
  }

  MachLin::Backw(lrate, wdecay, eff_bsize);
}
```

B.50 MachTanh.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTanh.h,v 1.7 2010/01/25 12:27:07 schwenk Exp $
 *
 * sigmoidal machine:  output = tanh(weights * input + biases)
 */
```

```
#ifndef _MachTanh_h
#define _MachTanh_h

#include "MachLin.h"

class MachTanh : public MachLin
{
public:
  MachTanh(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~MachTanh();
  virtual int GetMType() {return file_header_mtype_tanh;};  // get type
of machine
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
};

#endif
```

B.51 NBest.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NBest.cpp,v 1.8 2010/01/25 12:27:07 schwenk Exp schwenk $
 */


#include "NBest.h"
#include "NbestCSLM.h"
```

```
#include "Tools.h"

#include <sstream>
#include <algorithm>


bool NBest::ParseLine(inputfilestream &inpf, const int n) {
  static string line; // used internally to buffer an input line
  static int prev_id=-1; // used to detect a change of the n-best ID
  int id;
  vector<float> f;
  vector<string> blocks;


  if (line.empty()) {
    getline(inpf,line);
    if (inpf.eof()) return false;
  }

    // split line into blocks
  //cerr << "PARSE line: " << line << endl;
  int pos=0, epos;
  while ((epos=line.find(NBEST_DELIM,pos))!=string::npos) {
    blocks.push_back(line.substr(pos,epos-pos));
    // cerr << " block: " << blocks.back() << endl;
    pos=epos+strlen(NBEST_DELIM);
  }
  blocks.push_back(line.substr(pos,line.size()));
  // cerr << " block: " << blocks.back() << endl;

  if (blocks.size()<4) {
    cerr << "ERROR: can't parse the following line (skipped)" << endl
<< line << endl;
    line.clear(); // force read of new line
    return true;
  }

    // parse ID
  id=Scan<int>(blocks[0]);
  if (prev_id>=0 && id!=prev_id) {prev_id=id; return false;} // new
nbest list has started
  prev_id=id;
  //cerr << "same ID " << id << endl;

  if (n>0 && nbest.size() >= (uint) n) {
    //cerr << "skipped" << endl;
    line.clear();
    return true; // skip parsing of unused hypos
  }

    // parse feature function scores
  //cerr << "PARSE features: '" << blocks[2] << "' size: " <<
blocks[2].size() << endl;
  pos=blocks[2].find_first_not_of(' ');
  while (pos<blocks[2].size() && (epos=blocks[2].find("
",pos))!=string::npos) {
    string feat=blocks[2].substr(pos,epos-pos);
```

```
    //cerr << " feat: '" << feat << "', pos: " << pos << ", " << epos
<< endl;
    if (feat.find(":",0)!=string::npos) {
      //cerr << "  name: " << feat << endl;
    }
    else {
      f.push_back(Scan<float>(feat));
      //cerr << "  value: " << f.back() << endl;
    }
    pos=epos+1;
  }

    // eventually parse segmentation
  if (blocks.size()>4) {
    static int info=0;
    if (!info) cerr << " - skipping segmentation information" << endl;
    info=true;
    //Error("parsing segmentation not yet supported");
  }

  nbest.push_back(Hypo(id, blocks[1], f, Scan<float>(blocks[3])));

  line.clear(); // force read of new line

  return true;
}


NBest::NBest(inputfilestream &inpf, const int n) {
  //cerr << "NBEST: constructor with file called" << endl;
  while (ParseLine(inpf,n));
  //cerr << "NBEST: found " << nbest.size() << " lines" << endl;
}


NBest::~NBest() {
  //cerr << "NBEST: destructor called" << endl;
}

void NBest::Write(outputfilestream &outf, int n)
{
  if (n<1 || (uint) n>nbest.size()) n=nbest.size();
  for (int i=0; i<n; i++) nbest[i].Write(outf);
}


void NBest::CalcGlobal(Weights &w)
{
  //cerr << "NBEST: calc global of size " << nbest.size() << endl;
  for (vector<Hypo>::iterator i = nbest.begin(); i != nbest.end(); i++)
{
    (*i).CalcGlobal(w);
  }
}


void NBest::Sort() {
```

```
  sort(nbest.begin(),nbest.end());
}


void NBest::AddID(const int o)
{
  for (vector<Hypo>::iterator i = nbest.begin(); i != nbest.end(); i++)
{
    (*i).AddID(o);
  }
}

void NBest::RescoreLM(NbestLM &lm, const int lm_pos)
{
  for (vector<Hypo>::iterator i = nbest.begin(); i != nbest.end(); i++)
{
    lm.RescoreHyp(*i,lm_pos);
  }
  lm.FinishPending();
}
```

B.52 Nbest.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NBest.h,v 1.6 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _NBEST_H_
#define _NBEST_H_

using namespace std;
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

#include "Toolsgz.h"
#include "Hypo.h"
#include "NbestLM.h"

class NBest {
  int                id;
  string           src;
  vector<Hypo> nbest;
  bool ParseLine(inputfilestream &inpf, const int n);
 public:
  NBest(inputfilestream&, const int=0);
  ~NBest();
  int NbNBest() {return nbest.size(); };
  void CalcGlobal(Weights&);
  void Sort(); // largest values first
  void Write(outputfilestream&, int=0);
  void AddID(const int offs);
  void RescoreLM(NbestLM&, const int);
};


#endif
```

B.53 nbest_cmd.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

```
 *
 * $Id: nbest_cmd.cpp,v 1.8 2010/01/28 09:08:40 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <fstream>
#include <getopt.h>

#include "NBest.h"
#include "NbestLMSRI.h"
#include "NbestCSLM.h"


static struct option long_options[] =
  {
    {"input-file",1,0,'i'},
    {"inn",1,0,'I'},
    {"output-file",1,0,'o'},
    {"outn",1,0,'O'},
    {"offs",1,0,'a'},
    {"lm",1,0,'l'},
    {"order",1,0,'L'},
    {"cslm",1,0,'c'},
    {"weights",1,0,'w'},
    {"recalc",0,0,'r'},
    {"sort",0,0,'s'},
    {"lexical",0,0,'h'},
    {0, 0, 0, 0}
  };
int option_index;

void usage (bool do_exit=true)
{
   cout <<  "nbest_tool - A tool to process Moses n-best lists" << endl
      << "Copyright (C) 2010 Holger Schwenk, University of Le Mans,
France" << endl << endl;

#if 0
      << "This library is free software; you can redistribute it
and/or" << endl
      << "modify it under the terms of the GNU General Public" << endl
      << "License as published by the Free Software Foundation; either"
<< endl
      << "version 2.1 of the License, or (at your option) any later
version." << endl << endl

      << "This library is distributed in the hope that it will be
useful," << endl
      << "but WITHOUT ANY WARRANTY; without even the implied warranty
of" << endl
      << "MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU" << endl
      << "Lesser General Public License for more details." << endl <<
endl
```

```
      << "You should have received a copy of the GNU General Public" <<
endl
      << "License along with this library; if not, write to the Free
Software" << endl
      << "Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
02110-1301  USA" << endl << endl
      <<
"*****************************************************************
*" << endl << endl
      << "Built on " << __DATE__ << endl << endl;
#endif

  cout << "--input-file   -i  file name of the input n-best list" <<
endl;
  cout << "--inn          -I  number of hypothesis to read per n-best
(default all)" << endl;
  cout << "--output-file  -o  file name of the output n-best list" <<
endl;
  cout << "--outn         -O  number of hypothesis to write per n-best
(default all)" << endl;
  cout << "--offs         -a  add offset to n-best ID (useful for
seperately generated n-bests)" << endl;
  cout << "--lm           -l  rescore with a SRILM" << endl;
  cout << "--order        -L  order of the SRILM (default 4)" << endl;
  cout << "--cslm         -c  rescore with a CSLM" << endl;
  cout << "--vocab        -v  vocabulary to use with the CSLM" << endl;
  cout << "--lm-pos       -p  position of LM score (0 means to append
it)" << endl;
  cout << "--recalc       -r  recalc global scores" << endl;
  cout << "--weights      -w  coefficients of the feature functions" <<
endl;
  cout << "--sort         -s  sort n-best list according to the global
scores" << endl;
  cout << "--lexical      -d  report number of lexically different
hypothesis" << endl;

  if (do_exit) exit(1);
}

int main (int argc, char *argv[]) {

    // parse parameters
  string ifname, ofname, wfname, lmfname, cslmfname, vocab_fname;
  int in_n=0, out_n=0, offs=0, lm_pos=0, lm_order=4;
  bool do_lm=false, do_cslm=false, do_calc=false, do_sort=false,
do_lexical=false;
  char c;

  while ((c=getopt_long (argc, argv, "i:I:o:O:a:l:L:p:c:v:w:rsd",
         long_options, &option_index)) != -1) {
    switch (c) {
    case 'i':
      ifname = string(optarg);
      break;
    case 'I':
      in_n = strtol(optarg, NULL, 10);
      break;
```

```
    case 'o':
      ofname = string(optarg);
      break;
    case 'O':
      out_n = strtol(optarg, NULL, 10);
      break;
    case 'a':
      offs = strtol(optarg, NULL, 10);
      break;
    case 'l':
      lmfname = string(optarg);
      do_lm=true;
      break;
    case 'L':
      lm_order = strtol(optarg, NULL, 10);
      do_lm=true;
      break;
    case 'p':
      lm_pos = strtol(optarg, NULL, 10);
      break;
    case 'c':
      cslmfname = string(optarg);
      do_cslm=true;
      break;
    case 'v':
      vocab_fname = string(optarg);
      break;
    case 'w':
      wfname = string(optarg);
      break;
    case 'r':
      do_calc = true;
      break;
    case 's':
      do_sort = true;
      break;
    case 'h':
      do_lexical = true;
      break;
    default:
      usage();
    }
  }

  if (ifname.empty() || ofname.empty()) {
    usage(false);
    Error("\ninput-file and output files are required");
  }

    // read input

  cout << "NBest version 1.0, written by Holger.Schwenk@lium.univ-
lemans.fr" << endl
       << " - reading input from file '" << ifname << "'";
  if (in_n>0) cout << " (limited to the first " << in_n << "
hypothesis)";
  cout << endl;
```

```
  inputfilestream inpf(ifname.c_str());

    // open output
  cout << " - writing output to file '" << ofname << "'";
  if (out_n>0) cout << " (limited to the first " << out_n << "
hypothesis)";
  cout << endl;
  outputfilestream outf(ofname.c_str());

    // shall we add an offset to the ID ?
  if (offs>0)
    cout << " - adding offset of " << offs << " to the n-best ids" <<
endl;

    // shall we rescore with an LM ?
  NbestLMSRI lm;
  if (do_lm) {
    cout << " - rescoring with a " << lm_order << "-gram LM " <<
lmfname;
    if (lm_pos>0) cout << ", scores at position " << lm_pos;
             else cout << ", scores are appended";
    cout << endl;
    lm.Read(lmfname, lm_order);
  }

    // shall we rescore with an CSLM ?

  NbestCSLM cslm;
  if (do_cslm) {
    if (vocab_fname.empty())
      Error("You need to specify a vocabulary when rescoring with a
CSLM\n");
    cout << " - rescoring with CSLM " << cslmfname;
    if (lm_pos>0) cout << ", scores at position " << lm_pos;
             else cout << ", scores are appended";
    cout << endl;
    cslm.Read(cslmfname, vocab_fname);
  }


    // eventually read weights
  Weights w;
  if (!wfname.empty()) {
    cout << " - reading weights from file '" << wfname << "'";
    int n=w.Read(wfname.c_str());
    cout << " (found " << n << " values)" << endl;
  }

  if (do_calc) cout << " - recalculating global scores" << endl;

    // shall we sort ?
  if (do_sort) cout << " - sorting global scores" << endl;

   // main loop
  int nb_sent=0, nb_nbest=0;
  while (!inpf.eof()) {
    NBest nbest(inpf, in_n);
```

```
    if (nbest.NbNBest()>0) {
      if (offs!=0) nbest.AddID(offs);
      if (do_calc) nbest.CalcGlobal(w);
      if (do_sort) nbest.Sort();
      if (do_lm) nbest.RescoreLM(lm,lm_pos);
      if (do_cslm) nbest.RescoreLM(cslm,lm_pos);
      nbest.Write(outf, out_n);

      nb_sent++;
      nb_nbest+=nbest.NbNBest();
    }
  }
  inpf.close();
  outf.close();

    // display final statistics
  cout << " - processed " << nb_nbest << " n-best hypotheses in " <<
nb_sent << " sentences"
       << " (average " << (float) nb_nbest/nb_sent << ")" << endl;

  return 0;
}
```

B.54 NbestCSLM.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NbestCSLM.cpp,v 1.6 2010/01/28 09:08:40 schwenk Exp $
 */


#include "Ngram.h"      // from SRI, subclass of LM
```

```
#include "Hypo.h"
#include "NbestCSLM.h"

NbestCSLM::NbestCSLM()
 : mach(NULL), eval(NULL)
{
}

NbestCSLM::~NbestCSLM() {
  if (mach) delete mach;
  if (sri_vocab) delete sri_vocab;
  if (eval) delete eval;
}

bool NbestCSLM::Read (const string &fname, const string &vocab_fname)
{
  ifstream ifs;
  ifs.open(fname.c_str(),ios::binary);
  CHECK_FILE(ifs,fname.c_str());
  mach = Mach::Read(ifs);
  ifs.close();
  mach->Info();
  lm_order = mach->GetIdim()+1;
  eval = new EvalNgramBin(*mach);
  sri_vocab = new Vocab();
  {
    File file(vocab_fname.c_str(), "r");
    sri_vocab->read(file);
    sri_vocab->remove("-pau-");
  }
  cout << " - using vocabulary with " << sri_vocab->numWords() << "
words\n";

  // TODO: try to check consistency of the vocabulary
  return true;
}

//
//
//
void NbestCSLM::RescoreHyp (Hypo &hyp, const int lm_pos)
{
  static TextStats tstats;
  static const int max_words=16384;
  static const int max_chars=max_words*16;
  static char str[max_chars];
  static VocabString vstr[max_words+1];

  strcpy(str,hyp.GetCstr()); // we need to copy since parseWords()
modifies the string
  int nw = sri_vocab->parseWords(str, vstr, max_words + 1);
  if (nw == max_words+1) Error("too many words in one hypothesis\n");

  WordID wid[nw+3];
  int b=0;
    // start sentence with BOS ?
  if (mode & RESCORE_MODE_BOS) wid[b++]=sri_vocab->ssIndex();
```

```
  sri_vocab->getIndices(vstr, (VocabIndex*) (wid+b), nw + 1, sri_vocab-
>unkIndex());
#ifdef DEBUG
  for (int i=0;i<nw; i++) printf(" %s[%d]", vstr[i], wid[i+b]);
cout<<endl;
#endif

    // end sentence with EOS ?
  nw += b;
  if (mode & RESCORE_MODE_EOS) wid[nw++]=sri_vocab->seIndex();

  float logP=0;
  if (nw<lm_order)
    logP = log10(eval->Eval(wid, nw, NULL)); // only one ngram that is
short than LM order
  else {
    WordID *wptr=wid;
    int i=lm_order;
    while (i<=nw) {
      logP += log10(eval->Eval(wptr, lm_order, NULL));
      wptr++; i++;
    }
  }

  hyp.SetFeature(logP,lm_pos);
  return;
}

void NbestCSLM::FinishPending()
{
  //eval->BlockFinish();
}

//
//
//
float NbestCSLM::GetValue ()
{
  return 0;
}
```

B.55 NbestCSLM.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
```

```
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NbestCSLM.h,v 1.5 2010/01/28 09:08:40 schwenk Exp $
 */


#ifndef _NBESTCSLM_H_
#define _NBESTCSLM_H_

using namespace std;

#include "NbestLM.h"
#include "Vocab.h" // from the SRI toolkit
#include "Mach.h" // from the CSLM toolkit
#include "EvalNgramBin.h"

class NbestCSLM : public NbestLM {
protected:
  Vocab *sri_vocab;
  Mach *mach;
  int order;
  EvalNgramBin *eval;
public:
  NbestCSLM();
  virtual ~NbestCSLM();
  virtual float GetValue();
  virtual bool Read (const string &, const string&);
  virtual void RescoreHyp (Hypo &hyp, const int lm_pos);   // recalc LM
score on hypothesis, returns log10 probability
  virtual void FinishPending();
};

#endif
```

B.56 NbestLM.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
```

```
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NbestLM.cpp,v 1.6 2010/01/26 19:37:22 schwenk Exp $
 */

#include <iostream>
#include <stdlib.h>     // exit()
#include "NbestLM.h"

bool NbestLM::Read (const string &fname, int const order)
{
  cerr << "Read() of virtual class NbestLM called" << endl;
  exit(1);
  return false;
}
```

B.57 NbestLM.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
```

```
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NbestLM.h,v 1.6 2010/01/26 19:37:22 schwenk Exp $
 */

#ifndef _NBESTLM_H_
#define _NBESTLM_H_

using namespace std;

#include <string>
#include <vector>
#include "Hypo.h"

#define RESCORE_MODE_BOS 1
#define RESCORE_MODE_EOS 2

class NbestLM {
protected:
  string fname;  // translation
  int lm_order;  // order of NbestLM
  int mode;
  vector<int> nb_ngrams;  // nb of ngrams per order, nb_ngrams[0] is
voc. size
public:
  NbestLM() : mode(RESCORE_MODE_BOS | RESCORE_MODE_EOS) {};
  virtual ~NbestLM() {};
  virtual float GetValue() {return 0; };
  virtual bool Read (const string &, int const order = 4);
  virtual void RescoreHyp (Hypo &hyp, const int lm_pos) {};   // recalc
LM score on hypothesis
  virtual void FinishPending() {};  // finish pending requests, only
used for CSLM
};

#endif
```

B.58 NbestLMSRI.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
```

```
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NbestLMSRI.cpp,v 1.8 2010/01/26 19:33:05 schwenk Exp $
 */


#include <stdlib.h>     // exit()
#include "NbestLMSRI.h"
#include "Tools.h"

NbestLMSRI::NbestLMSRI()
 : sri_vocab(0), sri_ngram(0)
{
 //cerr << "NbestLMSRI::NbestLMSRI called" << endl;
}

NbestLMSRI::~NbestLMSRI() {
  delete sri_vocab;
  delete sri_ngram;
}

bool NbestLMSRI::Read (const string &fname, int const order) {

  //cout << " - reading SRI LM from file " << fname << endl;
  sri_vocab = new Vocab();
  sri_ngram = new Ngram(*sri_vocab, order);

    // reading LM
  lm_order = order;
  sri_ngram->setorder(lm_order);
  sri_ngram->skipOOVs() = false;
  File ngram_file(fname.c_str(), "r");
  sri_ngram->read(ngram_file, 0);

  sri_idx_unk = sri_vocab->unkIndex();
  sri_idx_bos = sri_vocab->ssIndex();
  sri_idx_eos = sri_vocab->seIndex();

    // get order and number of n-grams
  nb_ngrams.push_back(sri_vocab->numWords());
  cout << "   vocabulary: " << nb_ngrams[0] << " words; ngrams:";
  for (int o=1; o<=lm_order; o++) {
    nb_ngrams.push_back(sri_ngram->numNgrams(o));
    cout << " " << nb_ngrams.back();
    //if (nb_ngrams[o]==0) {order=o-1; break; };
  }
  cout << endl;

  return true;
}
```

```
//
//
//
void NbestLMSRI::RescoreHyp (Hypo &hyp, const int lm_pos)
{
  static TextStats tstats;
  static const int max_words=16384;
  static const int max_chars=max_words*16;
  static char str[max_chars];
  static VocabString vstr[max_words+1];

  if (mode != (RESCORE_MODE_BOS | RESCORE_MODE_EOS)) {
    fprintf(stderr,"ERROR: mode is set to %d, but the SRILM
automatically surrounds the sentence with <s> and </s>", mode);
    Error();
  }

  strcpy(str,hyp.GetCstr()); // we need to copy since parseWords()
modifies the string
  int nw = sri_vocab->parseWords(str, vstr, max_words + 1);
  if (nw == max_words+1) Error("too many words in one hypothesis\n");

  float logP = sri_ngram->sentenceProb(vstr, tstats);
  hyp.SetFeature(logP,lm_pos);
  return;
}
```

B.59 NbestLMSRI.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NbestLMSRI.h,v 1.6 2010/01/26 19:33:05 schwenk Exp $
 */
```

```
#ifndef _NBESTLMSRI_H_
#define _NBESTLMSRI_H_

using namespace std;

#include <string>
#include <vector>
#include "NbestLM.h"

// from the SRI toolkit
#include "Vocab.h"
#include "Ngram.h"


class NbestLMSRI : public NbestLM {
protected:
  Vocab *sri_vocab;     // SRI vocabulary
  Ngram *sri_ngram;     // pointer on SRI model
  VocabIndex sri_idx_unk, sri_idx_bos, sri_idx_eos;
public:
  NbestLMSRI(); // : sri_vocab(0), sri_ngram(0) {};
  virtual ~NbestLMSRI();
  virtual float GetValue() {return 0; };
  virtual bool Read (const string &, int const order = 4);
  virtual void RescoreHyp (Hypo &hyp, const int lm_pos);   // recalc
log10 LM score on hypothesis
};

#endif
```

B.60 net_info.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
```

```
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: net_info.cpp,v 1.4 2010/01/25 12:53:31 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include "Mach.h"

int main (int argc, char *argv[])
{
  ifstream ifs;
  Mach *m;

  for (int i=1; i<argc; i++) {
    ifs.open(argv[i],ios::binary);
    CHECK_FILE(ifs,argv[i]);
    cout << "\nInformation on machine: " << argv[i] << endl;
    m = Mach::Read(ifs);
    m->Info();
    ifs.close();
    delete m;
  }

  return 0;
}
```

B.61text2bin.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: text2bin.cpp,v 1.5 2010/01/25 12:27:07 schwenk Exp $
```

```
 *
 * text2bin.cpp: tool to convert UTF8 texts to binary representation
for the
 * CSLM toolkit (index in word list according to SRILM representation
of the
 * vocabulary)
 *
 * The previous program convertToInt sorts the word list using native
byte
 * values for the order. This can be reproduced by UNIX sort by setting
 * LC_ALL=C
 */

using namespace std;
#include <vector>

#include "File.h" // tool from SRILM for file IO
#include "Vocab.h"      // SRILM vocabulary representation

#define LINE_LEN 16384
#define WordID    int
#define HEADER_LEN (4*sizeof(int)+3*sizeof(WordID))

#undef COUNT_OOV        // this needs to be debugged first

int main (int argc, char *argv[]) {
  Vocab *voc = new Vocab;
  Vocab *oov_w = new Vocab;
  int i, nvoc;
  vector<int> oov_cnt;
  WordID idx;
  char      line[LINE_LEN];

  cout << "Text to binary converter V1.0, H. Schwenk, LIUM, University
of Le Mans, France" << endl;
    // parse args
  if (argc!=5) {
    cerr << " usage: " << argv[0] << " input-vocab output-binary-file
output-word-freq output-list-of-oov < file" << endl;
    return 1;
  }
  char *voc_fname=argv[1];
  char *bin_fname=argv[2];
  char *wfreq_fname=argv[3];
  char *oov_fname=argv[4];

  voc->unkIsWord() = true;
  voc->toLower() = false;

    // read vocabulary
  {
    File file(voc_fname, "r");
    voc->read(file);
    voc->remove("-pau-");
  }
  nvoc=voc->numWords(); // SRILM may add <unk>, <s> or </s> to the
provided word list !!
```

```
  WordID idx_unk=voc->getIndex(Vocab_Unknown);
  WordID idx_bos=voc->getIndex(Vocab_SentStart);
  WordID idx_eos=voc->getIndex(Vocab_SentEnd);
  printf(" - using word list %s (%d words, unk=%d, bos=%d, eos=%d)\n",
voc_fname, nvoc, idx_unk, idx_bos, idx_eos);

  int *wordfreq = new int[nvoc+1];  // SRILM returns values 1..nvoc !!
  for (i=0; i<nvoc; i++) wordfreq[i]=0;
#ifdef COUNT_OOV
  oov_cnt.resize(nvoc);
  for (i=0; i<nvoc; i++) { oov_cnt[i]=wordfreq[i]=0; }
#endif

    // write dummy header
  cout << " - writing binary representation to file " << bin_fname <<
endl;
  File binf(bin_fname, "wb");
  char      header[HEADER_LEN];
  fwrite(&header, sizeof(char), HEADER_LEN, binf);


    // read file, convert to binary, count word frequencies and #unk
  int nl=0, nw=0, nunk=0;
  while (cin.getline(line, LINE_LEN)) {
    line[strlen(line)]=0;
    line[strlen(line)+1]=0;
    nl++;
    fwrite(&idx_bos, sizeof(idx_bos), 1, binf);

    char    *bptr = line, *eptr;

    while ((*bptr != 0) && (*bptr != '\n') && (*bptr == ' ')) bptr++;
/* skip blank */
    if (*bptr == '\n') continue;    /* skip empty lines */

      // loop on all words in line
    //cerr << "Line: " << line << endl;
    while ((*bptr != 0) && (*bptr != '\n')) {
      //cin >> w;

      eptr = bptr + 1;
      while ((*eptr != 0) && (*eptr != '\n') && (*eptr != ' ')) eptr++;
      *eptr = 0;

      idx = voc->getIndex(bptr);
      //cerr << bptr << "[" << idx <<"]"<< endl;
      nw++;
      if (nw%1000000 == 0) cout << "\r - processing " << nw/1000000 <<
"M words";
      if (idx==(WordID) Vocab_None) {
        fwrite(&idx_unk, sizeof(idx_unk), 1, binf);
        nunk++;
        idx=oov_w->addWord(bptr);
#ifdef COUNT_OOV
        if (idx<0) {
          fprintf(stderr,"illegal OOV idx (%d) for word %s\n",idx,
bptr);
```

```
          exit(1);
        }
        if (idx>oov_cnt.capacity())
oov_cnt.reserve(2*oov_cnt.capacity());
        oov_cnt[idx]++;  // TODO: resize vector ??
#endif
      }
      else {
        fwrite(&idx, sizeof(idx), 1, binf);
        if (idx<1 || idx>nvoc) {
          fprintf(stderr,"illegal word index (%d) for word %s\n",idx,
bptr);
          exit(1);
        }
        wordfreq[idx]++;
      }

      bptr = eptr + 1;
      while ((*bptr != 0) && (*bptr != '\n') && (*bptr == ' ')) bptr++;

    }
    fwrite(&idx_eos, sizeof(idx_eos), 1, binf);
    for (i=0; i<LINE_LEN; i++) line[i]=0; // we need to clear the
buffer !?
  }
  cout << "\r";

    // dump vocabulary with word frequencies to file
  int ndiff=0;
  {
    cout << " - dumping word frequencies to file " << wfreq_fname <<
endl;
    File file(wfreq_fname, "w");
    for (i=0; i<nvoc; i++) {
      if (wordfreq[i]) ndiff++;
      fprintf(file, "%s %d\n", voc->getWord(i), wordfreq[i]);
    }
  }

    // dump list of OOVs to file
  {
    cout << " - dumping list of OOV to file " << oov_fname << endl;
    File file(oov_fname, "w");
    oov_w->remove("-pau-");
    oov_w->remove("<unk>");
    oov_w->remove("<s>");
    oov_w->remove("</s>");
    for (i=0; i<(WordID) oov_w->numWords(); i++) {
      if (oov_w->getWord(i))
#ifdef COUNT_OOV
        fprintf(file, "%s %d\n", oov_w->getWord(i), oov_cnt[i]);
#else
        fprintf(file, "%s\n", oov_w->getWord(i));
#endif
    }
  }
```

```
    // write header with actual values: nb_lines, nb_words, nbvoc, bos,
eos, unk
  rewind(binf);
  fwrite(&nl, sizeof(int), 1, binf);
  fwrite(&nw, sizeof(int), 1, binf);
  fwrite(&nvoc, sizeof(int), 1, binf);
  i=sizeof(WordID);
  fwrite(&i, sizeof(int), 1, binf);
  fwrite(&idx_bos, sizeof(WordID), 1, binf);
  fwrite(&idx_eos, sizeof(WordID), 1, binf);
  fwrite(&idx_unk, sizeof(WordID), 1, binf);

    // print final stats
  printf(" - %d lines with %d words processed, %d uniq words (%5.2f%%
of the vocabulary)\n",
      nl, nw, ndiff, 100.0*ndiff/nvoc);
  printf(" - %d words were unknown (%5.2f%% of the text), %d new
words\n", nunk, 100.0*nunk/nw, oov_w->numWords()-4);
    // we remove the 4 spezial words from the UNK vocab, but the size
of the vocabulary
    // is not corrected by the SRI toolkit

  delete [] wordfreq;
  delete voc;
  delete oov_w;
#ifdef COUNT_OOV
  oov_cnt.clear();
#endif

  return 0;
}
```

B.62 Tools.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Tools.cpp,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"

void Error(void)
{
  exit(1);
}

void Error(const char *txt)
{
  cerr << "ERROR: " << txt << endl;
  exit(1);
}

int ReadInt(ifstream &inpf, const string &name, int minval,int maxval)
{
  string buf;
  inpf >> buf;
  if (buf!=name) {
    cerr << "FileRead: found field '" << buf << "' while looking for '"
<< name << "'";
    Error("");
  }

  int val;
  inpf >> val;
  if (val<minval || val>maxval) {
    cerr << "FileRead: values for " << name << "must be in
["<<minval<<","<<maxval<<"]";
    Error("");
  }

  return val;
}
```

B.63 Tools.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
```

```
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Tools.h,v 1.15 2010/01/25 12:53:31 schwenk Exp $
 */

#ifndef _Tools_h
#define _Tools_h

#include <iostream>
#include <fstream>
#include <string.h>     // memcpy()
#include <stdlib.h>     // exit()
#include <math.h>

typedef float REAL;
typedef unsigned int uint;
typedef long unsigned int luint;

//
// general purpose helper functions
//
#ifdef DEBUG
# define TRACE(txt) cout << txt;
# define debug(F) printf(F)
# define debug1(F,a) printf(F,a)
# define debug2(F,a,b) printf(F,a,b)
# define debug3(F,a,b,c) printf(F,a,b,c)
# define debug4(F,a,b,c,d) printf(F,a,b,c,d)
# define debug5(F,a,b,c,d,e) printf(F,a,b,c,d,e)
# define debug6(F,a,b,c,d,e,f) printf(F,a,b,c,d,e,f)
# define debug7(F,a,b,c,d,e,f,h) printf(F,a,b,c,d,e,f,h)
# define debug8(F,a,b,c,d,e,f,h,i) printf(F,a,b,c,d,e,f,h,i)
#else
# define TRACE(txt)
# define debug(F)
# define debug1(F,a)
# define debug2(F,a,b)
# define debug3(F,a,b,c)
# define debug4(F,a,b,c,d)
# define debug5(F,a,b,c,d,e)
# define debug6(F,a,b,c,d,e,f)
# define debug7(F,a,b,c,d,e,f,h)
# define debug8(F,a,b,c,d,e,f,h,i)
#endif
```

```
void Error(void);
void Error(const char *txt);

#define CHECK_FILE(ifs,fname) if(!ifs) { perror(fname); Error(); }

//
// parsing of ASCII files
//
int ReadInt(ifstream&,const string&,int=0,int=2147483647); // TODO:
MAXINT
float ReadFloat(ifstream&,const string&,float=0,float=3.4e38); // TODO:
MAXFLOAT
string ReadText(ifstream&,const string&);

#endif
```

B.64 Toolsgz.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Toolsgz.cpp,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <stdexcept>
#include <stdlib.h>
#include "Toolsgz.h"

int Weights::Read(const char *fname) {
  ifstream inpf;

  inpf.open(fname);
```

```
  if (inpf.fail()) {
    perror ("ERROR"); exit(1);
  }

  float f;
  while (inpf >> f) val.push_back(f);

  inpf.close();
  return val.size();
}

//
//
//

void gzifstream::open(char *fname)
{
  //check if file is readable
  std::filebuf* fb = new std::filebuf();
  _fail=(fb->open(fname, std::ios::in)==NULL);

  char *sptr=strrchr(fname,'.');
  if (sptr && strcmp(sptr,".gz")==0) {
    fb->close(); delete fb;
    gz_streambuf = new gzfilebuf(fname);
  } else {
    gz_streambuf = fb;
  }
  this->init(gz_streambuf);
}


void gzofstream::open(char *fname)
{
  //check if file is readable
  std::filebuf* fb = new std::filebuf();
  _fail=(fb->open(fname, std::ios::out)==NULL);
  //cerr << "fail: " << _fail <<endl;

  char *sptr=strrchr(fname,'.');
  if (sptr && strcmp(sptr,".gz")==0) {
    fb->close(); delete fb;
    gz_streambuf = new gzfilebuf(fname);
  } else {
    gz_streambuf = fb;
  }
  this->init(gz_streambuf);
}

//
//
//

inputfilestream::inputfilestream(const std::string &filePath)
: std::istream(0),
m_streambuf(0)
{
```

```
  //check if file is readable
  std::filebuf* fb = new std::filebuf();
  _good=(fb->open(filePath.c_str(), std::ios::in)!=NULL);

  if (filePath.size() > 3 &&
      filePath.substr(filePath.size() - 3, 3) == ".gz")
  {
    fb->close(); delete fb;
    m_streambuf = new gzfilebuf(filePath.c_str());
  } else {
    m_streambuf = fb;
  }
  this->init(m_streambuf);
}

inputfilestream::~inputfilestream()
{
  delete m_streambuf; m_streambuf = 0;
}

void inputfilestream::close()
{
}

outputfilestream::outputfilestream(const std::string &filePath)
: std::ostream(0),
m_streambuf(0)
{
  //check if file is readable
  std::filebuf* fb = new std::filebuf();
  _good=(fb->open(filePath.c_str(), std::ios::out)!=NULL);

  if (filePath.size() > 3 && filePath.substr(filePath.size() - 3, 3) ==
".gz")
  {
    fb->close(); delete fb;
    m_streambuf = new gzfilebuf(filePath.c_str());
  } else {
    m_streambuf = fb;
  }
  this->init(m_streambuf);
}

outputfilestream::~outputfilestream()
{
  delete m_streambuf; m_streambuf = 0;
}

void outputfilestream::close()
{
}
```

B.65 Toolsgz.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Toolsgz.h,v 1.4 2010/01/26 19:43:09 schwenk Exp $
 */

#ifndef TOOLSGZ_H
#define TOOLSGZ_H

using namespace std;

#include <stdexcept>
#include <limits>
#include <cstring>      // for memmove
#include <zlib.h>
#include <vector>
#include <map>
#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <streambuf>

#define US_NOSET (numeric_limits<unsigned short>::max())
#define MAX_LINE  1024


//
//
//

class Weights {
  vector<float> val;
```

```
 public:
  Weights() {};
  ~Weights() {};
  int Read(const char *);
  friend class Hypo;
};


class gzfilebuf : public std::streambuf {
public:
  gzfilebuf(const char *filename)
  { _gzf = gzopen(filename, "rb");
    setg (_buff+sizeof(int),     // beginning of putback area
          _buff+sizeof(int),     // read position
          _buff+sizeof(int));    // end position
  }
  gzfilebuf(const char *filename, int dummy)
  { _gzf = gzopen(filename, "w+b");
    setg (_buff+sizeof(int),     // beginning of putback area
          _buff+sizeof(int),     // read position
          _buff+sizeof(int));    // end position
  }
  ~gzfilebuf() { gzclose(_gzf); }
protected:
  virtual int_type overflow (int_type c) {
      throw;
  }

  // write multiple characters
  virtual
  std::streamsize xsputn (const char* s,
                          std::streamsize num) {
      throw;
  }

  virtual std::streampos seekpos ( std::streampos sp,
std::ios_base::openmode which = std::ios_base::in | std::ios_base::out
){ throw;
  }

  //read one character
  virtual int_type underflow () {
    // is read position before end of _buff?
        if (gptr() < egptr()) {
            return traits_type::to_int_type(*gptr());
        }

        /* process size of putback area
         * - use number of characters read
         * - but at most four
         */
        unsigned int numPutback = gptr() - eback();
        if (numPutback > sizeof(int)) {
            numPutback = sizeof(int);
        }

        /* copy up to four characters previously read into
```

```
         * the putback _buff (area of first four characters)
         */
        memmove (_buff+(sizeof(int)-numPutback), gptr()-numPutback,
                      numPutback);

        // read new characters
        int num = gzread(_gzf, _buff+sizeof(int), _buffsize-
sizeof(int));
        if (num <= 0) {
            // ERROR or EOF
            return EOF;
        }

        // reset _buff pointers
        setg (_buff+(sizeof(int)-numPutback),   // beginning of putback
area
              _buff+sizeof(int),                // read position
              _buff+sizeof(int)+num);           // end of buffer

        // return next character
        return traits_type::to_int_type(*gptr());
  }

  std::streamsize xsgetn (char* s,
                          std::streamsize num) {
    return gzread(_gzf,s,num);
  }

private:
  gzFile _gzf;
  static const unsigned int _buffsize = 1024;
  char _buff[_buffsize];
};

//
//
//

class inputfilestream : public std::istream
{
protected:
        std::streambuf *m_streambuf;
      bool _good;
public:

        inputfilestream(const std::string &filePath);
        ~inputfilestream();
      bool good(){return _good;}
        void close();
};

class outputfilestream : public std::ostream
{
protected:
        std::streambuf *m_streambuf;
      bool _good;
public:
```

```
        outputfilestream(const std::string &filePath);
        ~outputfilestream();
      bool good(){return _good;}
        void close();
};

/***************************************************
 *
 * Compressed File IO
 */

class gzifstream : public std::istream
{
protected:
  std::streambuf *gz_streambuf;
  bool _fail;
public:
  gzifstream() : gz_streambuf(0), _fail(true) {};
  ~gzifstream() {if (gz_streambuf) delete(gz_streambuf); };
  void open(char*);
  bool fail() {return _fail;}
  void close() {};
};


class gzofstream : public std::ostream
{
protected:
  std::streambuf *gz_streambuf;
  bool _fail;
public:
  gzofstream() : gz_streambuf(0), _fail(true) {};
  ~gzofstream() {if (gz_streambuf) delete(gz_streambuf); };
  void open(char*);
  bool fail() {return _fail;}
  void close() {};
};


//***************************************************


template<typename T>
inline T Scan(const std::string &input)
{
         std::stringstream stream(input);
         T ret;
         stream >> ret;
         return ret;
}


#endif
```

B.66 Trainer.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Trainer.cpp,v 1.9 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "Mach.h"
#include "Trainer.h"

Trainer::Trainer (Mach *pmach, ErrFct *perrfct,
              char *train_fname, char *dev_fname,
              REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,
              int p_maxep, int p_ep)
 : mach(pmach), errfct(perrfct),
   lrate_beg(p_lr_beg), lrate_mult(p_lr_mult), wdecay(p_wd),
   nb_epoch(p_ep), max_epoch(p_maxep)
{
  char      msg[1024];

  idim=mach->GetIdim(); odim=mach->GetOdim(); bsize=mach->GetBsize();
  if (train_fname) {
    data_train = new Data(train_fname);

    if (idim != data_train->GetIdim()) {
      sprintf(msg,"Trainer: input dimension of the training data (%d)
does not match the one of the machine (%d)\n", data_train->GetIdim(),
idim);
```

```
      Error(msg);
    }
    if (odim != data_train->GetOdim()) {
      sprintf(msg,"Trainer: ouput dimension of the training data (%d)
does not match the one of the machine (%d)\n", data_train->GetOdim(),
odim);
      Error(msg);
    }
  }
  else
    data_train=NULL;

  if (dev_fname) {
    data_dev = new Data(dev_fname);
    if (idim != data_dev->GetIdim())
      Error("Trainer: input dimension of the validation data does not
match the one of the machine\n");
    if (odim != data_dev->GetOdim())
      Error("Trainer: output dimension of the validation data does not
match the one of the machine\n");
    }
  else
    data_dev=NULL;

  buf_input = new REAL[idim*bsize];
  buf_target = new REAL[odim*bsize];
  // memory for the output gradient is allocated by the error function
}

//*********************************************************************
*****************

Trainer::~Trainer ()
{
  if (data_train) delete data_train;
  if (data_dev) delete data_dev;
  delete [] buf_input;
  delete [] buf_target;
}

//*********************************************************************
*****************
// default lrate = mach->lrate_begin / (1.0 + total_n_ex_seen * mach-
>lrate_mult);
// default wdecay: constant

void Trainer::SetLrate()
{
  lrate = lrate_beg / (1.0 + mach->GetNbForw() * lrate_mult);
}

//*********************************************************************
*****************

REAL Trainer::Train()
{
#ifdef DEBUG
```

```
  printf("*****************\n");
  printf("Trainer::Train():\n");
  printf(" -  data_in: %p \n", (void*) buf_input);
  printf(" -   target: %p \n", (void*) buf_target);
  printf(" - grad_out: %p \n", (void*) errfct->GetGrad());
#endif
  data_train->Rewind();

  REAL err=0;
  nb_ex=0;
  mach->SetDataIn(buf_input);
  mach->SetGradOut(errfct->GetGrad());
  errfct->SetOutput(mach->GetDataOut());
  errfct->SetTarget(buf_target);
  bool data_available;
  do {
      // get a bunch of data
    int n=0;
    data_available = true;
    while (n < mach->GetBsize() && data_available) {
      data_available = data_train->Next();
      if (!data_available) break;
      memcpy(buf_input  + n*idim, data_train->input,
idim*sizeof(REAL));
      memcpy(buf_target + n*odim, data_train->target,
odim*sizeof(REAL));
      n++;
    }

    if (n>0) {
      mach->Forw(n);
      err += errfct->CalcGrad(n);
#ifdef DEBUG
      printf("OUTPUT:") ; for (int i=0;i<odim; i++) printf("
%4.1f",mach->GetDataOut()[i]); printf("\n");
      printf("TARGET:") ; for (int i=0;i<odim; i++) printf("
%4.1f",data_train->target[i]); printf("\n");
      printf("  GRAD:") ; for (int i=0;i<odim; i++) printf("
%4.1f",errfct->GetGrad()[i]); printf("\n");
#endif
      SetLrate();
      mach->Backw(lrate, wdecay, n);
    }

    nb_ex += n;
  } while (data_available);
  err /= nb_ex;

  return err;
}

//*********************************************************************
****************
// This should be overriden to do a task-specific validation

REAL Trainer::TestDev()
{
```

```
  if (!data_dev) return -1;

  int nb_ex_dev=0;
  REAL err=0;
  data_dev->Rewind();
  mach->SetDataIn(buf_input);
  errfct->SetOutput(mach->GetDataOut());
  errfct->SetTarget(buf_target);
  bool data_available;
  do {
      // get a bunch of data
    int n=0;
    data_available = true;
    while (n < mach->GetBsize() && data_available) {
      data_available = data_dev->Next();
      if (!data_available) break;
      memcpy(buf_input  + n*idim, data_dev->input, idim*sizeof(REAL));
      memcpy(buf_target + n*odim, data_dev->target, odim*sizeof(REAL));
      n++;
    }

      // process the bunch
    if (n>0) {
      mach->Forw(n);
      err += errfct->CalcValue(n);
#ifdef DEBUG
      printf(" INPUT:") ; for (int i=0;i<idim; i++) printf("
%4.1f",mach->GetDataIn()[i]); printf("\n");
      printf("OUTPUT:") ; for (int i=0;i<odim; i++) printf("
%4.1f",mach->GetDataOut()[i]); printf("\n");
      printf("TARGET:") ; for (int i=0;i<odim; i++) printf("
%4.1f",data_dev->target[i]); printf(" -> %f\n",errfct->CalcValue(n));
#endif
    }

    nb_ex_dev += n;
  } while (data_available);

  if (nb_ex_dev>0) return err/nb_ex_dev;
  return -1;
}


//*****************************************************************
*************
// simple training routine

void Trainer::TrainAndTest ()
{
  const int hlen=256;
  char hostname[hlen];
  gethostname(hostname, hlen); hostname[hlen-1]=0;
  cout << "Starting training on host " << hostname << " pid " <<
getpid() << endl;
  cout << " - training on " << data_train->GetFname() << endl;
  if (data_dev)
    cout << " - validation on " << data_dev->GetFname() << endl;
```

```
  cout << " - stopping training at " << max_epoch << " epochs" << endl;
  mach->Info();

  while (!Converged()) {
    InfoPre();
    err_train = Train();
    InfoPost();

    cout << " - starting validation ..."; cout.flush();
    err_dev = TestDev();
    if (err_dev<0)
      cout << " avrg error: no examples !?" << endl;
    else
      cout << " avrg error: " << err_dev << endl;
  }
  cout << "Training stopped" << endl;
  mach->Info();
  //mach->Write();
}

//************************************************************************
*****************

bool Trainer::Converged ()
{
  return (nb_epoch >= max_epoch);
}

//************************************************************************
*****************
// information before starting an epoch

void Trainer::InfoPre ()
{
  time_t now;
  time(&now); // TODO: ctime is not rentrant ! use ctime_r() instead if
needed
  cout << "Starting epoch " << ++nb_epoch << " at " << ctime(&now);

  SetLrate();
  fprintf(stdout, " - intial lrate=%6.4e, wdecay=%6.4e\n", lrate,
wdecay);
}

//************************************************************************
*****************
// information after finishing an epoch

void Trainer::InfoPost ()
{
  cout << " - epoch finished, " << nb_ex << " examples seen, average
error: " << err_train << endl;
}
```

B.67 Trainer.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Trainer.h,v 1.5 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Trainer_h
#define _Trainer_h

#include <iostream>
#include "Tools.h"
#include "Mach.h"
#include "ErrFct.h"
#include "Data.h"


class Trainer
{
private:
protected:
  Mach  *mach;                // network to train
  ErrFct *errfct;       // error fucntion to use
  Data  *data_train;          // training data to use
  Data  *data_dev;            // development data to use
    // buffer to store bsize examples
  REAL  *buf_input;
  REAL  *buf_target;
    // current learning rates
  REAL lrate_beg, lrate_mult; // params for exponential decay
  REAL lrate, wdecay;         // current values
    // stats
  int nb_ex;                  // during one epoch
  int nb_epoch;               // total nb of epochs
```

```
  int max_epoch;        // max numebr of epochs
  int idim, odim, bsize;      // copied here for faster access
  REAL err_train;       // average error during training
  REAL err_dev;               // average error during testing
   // internal helper functions
  virtual void SetLrate();          // modify learning rates
  virtual bool Converged();         // return TRUE if training has
converged or should be stopped
  virtual void InfoPre();           // dump information before starting
a new training epoch
  virtual void InfoPost();          // dump information after finishing
a training epoch
public:
  Trainer(Mach*, ErrFct*, char*, char* =NULL,   // mach, errfct, train,
dev
          float = 0.01, float =0, float =0,     // lrate_beg,
lrate_mult, wdecay
        int =10, int =0);                 // max epochs, current epoch
  virtual ~Trainer();
  virtual REAL Train();             // train for one epoch
  virtual REAL TestDev();           // test current network on dev data
                                // returns obtained error (-1 if error)
  virtual void TrainAndTest();            // main training routine for
X iterations
};

#endif
```

B.68 TrainerNgram.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: TrainerNgram.cpp,v 1.13 2010/01/25 12:27:07 schwenk Exp $
```

```
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "Mach.h"
#include "TrainerNgram.h"

TrainerNgram::TrainerNgram (Mach *pmach, ErrFct *perrfct,
      char *train_fname, char *dev_fname,
      REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,
      int p_maxep, int p_ep)
 :
Trainer(pmach,perrfct,NULL,NULL,p_lr_beg,p_lr_mult,p_wd,p_maxep,p_ep),
   order(0)
{
  char      msg[1024];

  idim=mach->GetIdim(); odim=mach->GetOdim(); bsize=mach->GetBsize();

  if (odim < 16) {
    sprintf(msg,"TrainerNgram: output dimension of the machine is
suspiciously small (%d)\n", odim);
    Error(msg);
  }

  if (train_fname) {
    data_train = new Data(train_fname);
    if (idim != data_train->GetIdim()) {
      sprintf(msg,"TrainerNgram: input dimension of the training data
(%d) does not match the one of the machine (%d)\n", data_train-
>GetIdim(), idim);
      Error(msg);
    }
    if (data_train->GetOdim() != 1) {
      sprintf(msg,"TrainerNgram: output dimension of the training data
should be 1, found %d\n", data_train->GetOdim());
      Error(msg);
    }
  }
  else
    data_train=NULL;

  if (dev_fname) {
    data_dev = new Data(dev_fname);
    if (idim != data_dev->GetIdim()) {
      sprintf(msg,"TrainerNgram: input dimension of the validation data
(%d) does not match the one of the machine (%d)\n", data_dev-
>GetIdim(), idim);
      Error(msg);
    }
    if (data_dev->GetOdim() != 1) {
      sprintf(msg,"TrainerNgram: output dimension of the validation
data should be 1, found %d\n", data_dev->GetOdim());
```

```
      Error(msg);
    }
  }
  else
    data_dev=NULL;
}

TrainerNgram::TrainerNgram (Mach *pmach, ErrFct *perrfct, Data &data)
 : Trainer(pmach,perrfct,NULL,NULL,0,0,0,0,0),
    order(0)
{
  char      msg[1024];

  idim=mach->GetIdim(); odim=mach->GetOdim(); bsize=mach->GetBsize();

  if (odim < 16) {
    sprintf(msg,"TrainerNgram: output dimension of the machine is
suspiciously small (%d)\n", odim);
    Error(msg);
  }

  data_train=NULL;
  data_dev=&data;

  if (idim != data_dev->GetIdim()) {
    sprintf(msg,"TrainerNgram: input dimension of the validation data
(%d) does not match the one of the machine (%d)\n", data_dev-
>GetIdim(), idim);
    Error(msg);
  }
  if (data_dev->GetOdim() != 1) {
    sprintf(msg,"TrainerNgram: output dimension of the validation data
should be 1, found %d\n", data_dev->GetOdim());
    Error(msg);
  }
}

//*********************************************************************
*****************
// default lrate = mach->lrate_begin / (1.0 + total_n_ex_seen * mach-
>lrate_mult);
// default wdecay: constant

void TrainerNgram::SetLrate()
{
  lrate = lrate_beg / (1.0 + mach->GetNbForw() * lrate_mult);
}

//*********************************************************************
*****************

REAL TrainerNgram::Train()
{
  if (!data_train) return -1;
#ifdef DEBUG
  printf("*****************\n");
  printf("TrainerNgram::Train():\n");
```

```
  printf(" -  data_in: %p \n", (void*) buf_input);
  printf(" -   target: %p \n", (void*) buf_target);
  printf(" - grad_out: %p \n", (void*) errfct->GetGrad());
#endif
  data_train->Rewind();

  REAL log_sum=0;
  nb_ex=0;
  mach->SetDataIn(buf_input);
  mach->SetGradOut(errfct->GetGrad());
  errfct->SetOutput(mach->GetDataOut());
  errfct->SetTarget(buf_target);
  bool data_available;
  do {
      // get a bunch of data
      // TODO: exlude out of slist
    int n=0;
    data_available = true;
    while (n < mach->GetBsize() && data_available) {
      data_available = data_train->Next();
      if (!data_available) break;
      memcpy(buf_input  + n*idim, data_train->input,
idim*sizeof(REAL));
      memcpy(buf_target + n*1, data_train->target, 1*sizeof(REAL));
      n++;
    }

    //if (nb_ex%1024==0) printf("."); fflush (stdout);

    if (n>0) {
      mach->Forw(n);
      log_sum += errfct->CalcGrad(n);
#ifdef DEBUG2
      int t=(int) data_train->target[0];
printf("OUTPUT:") ; for (int i=t-2;i<=t+2; i++) printf(" %f",mach-
>GetDataOut()[i]); printf("\n");
printf("TARGET:") ; for (int i=0;i<1; i++) printf(" %f",data_train-
>target[i]); printf("\n");
printf("  GRAD:") ; for (int i=t-2;i<=t+2; i++) printf(" %f",errfct-
>GetGrad()[i]); printf("\n");
#endif
      SetLrate();
      mach->Backw(lrate, wdecay, n);
    }

    nb_ex += n;
  } while (data_available);


  if (nb_ex>0) return exp(-log_sum / (REAL) nb_ex);  // return
perplexity

  return -1;
}

//*****************************************************************
****************
```

```
// This should be overriden to do a task-specific validation

REAL TrainerNgram::TestDev(char *fname)
{
  if (!data_dev) return -1;

  if (fname) {
    Error("not yet implemented");
  }

  int nb_ex_dev=0;
  REAL log_sum=0;
  data_dev->Rewind();
  mach->SetDataIn(buf_input);
  errfct->SetOutput(mach->GetDataOut());
  errfct->SetTarget(buf_target);
  bool data_available;
  do {
      // get a bunch of data
      // TODO: exlude out of slist
    int n=0;
    data_available = true;
    while (n < mach->GetBsize() && data_available) {
      data_available = data_dev->Next();
      if (!data_available) break;
      memcpy(buf_input  + n*idim, data_dev->input, idim*sizeof(REAL));
      memcpy(buf_target + n*1, data_dev->target, 1*sizeof(REAL));
      n++;
    }

      // process the bunch
    if (n>0) {
#ifdef DEBUG
printf("in:"); for (int i=0;i<idim;i++) printf(" %f", buf_input[i]);
printf("-> trg:"); for (int i=0;i<1;i++) printf(" %f", buf_target[i]);
printf("\n");
#endif
      mach->Forw(n);
      log_sum += errfct->CalcValue(n);
      if (fname) {
        Error();  // TODO: we should get access to the parts of bsize
      }
    }

    nb_ex_dev += n;
  } while (data_available);

  if (nb_ex_dev>0) return exp(-log_sum / (REAL) nb_ex_dev);  // return
perplexity
  return -1;
}


//*****************************************************************
*************
// simple training routine
```

```
void TrainerNgram::TrainAndTest ()
{
  if (!data_train) {
    cout << "No training data specified, training impossible" << endl;
    return;
  }

  const int hlen=256;
  char hostname[hlen];
  gethostname(hostname, hlen); hostname[hlen-1]=0;
  cout << "Starting training on host " << hostname << " pid " <<
getpid() << endl;
  cout << " - training on " << data_train->GetFname() << endl;
  if (data_dev)
    cout << " - validation on " << data_dev->GetFname() << endl;
  cout << " - stopping training at " << max_epoch << " epochs" << endl;
  mach->Info();

  while (!Converged()) {
    InfoPre();
    err_train = Train();
    InfoPost();

    cout << " - starting validation ..."; cout.flush();
    err_dev = TestDev();
    if (err_dev<0)
      cout << " avrg error: no examples !?" << endl;
    else
      cout << " avrg error: " << err_dev << endl;
  }
  cout << "Training stopped" << endl;
  mach->Info();
  //mach->Write();
}

//*****************************************************************
*****************

bool TrainerNgram::Converged ()
{
  return (nb_epoch >= max_epoch);
}

//*****************************************************************
*****************
// information before starting an epoch

void TrainerNgram::InfoPre ()
{
  time_t now;
  time(&now); // TODO: ctime is not rentrant ! use ctime_r() instead if
needed
  cout << "Starting epoch " << ++nb_epoch << " at " << ctime(&now);

  SetLrate();
  fprintf(stdout, " - intial lrate=%6.4e, wdecay=%6.4e\n", lrate,
wdecay);
```

```
}

//*****************************************************************
*****************
// information after finishing an epoch

void TrainerNgram::InfoPost ()
{
  cout << " - epoch finished, " << nb_ex << " examples seen, average
error: " << err_train << endl;
}
```

B.69 TrainerNgram.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: TrainerNgram.h,v 1.5 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _TrainerNgram_h
#define _TrainerNgram_h

#include <ostream>
#include "Tools.h"
#include "Mach.h"
#include "ErrFct.h"
#include "Data.h"
#include "Trainer.h"


class TrainerNgram : public Trainer
{
private:
```

```
    // copies of important fields
  int order;                  // from Data
protected:
   // internal helper functions
  virtual void SetLrate();          // modify learning rates
  virtual bool Converged();         // return TRUE if training has
converged or should be stopped
  virtual void InfoPre();           // dump information before starting
a new training epoch
  virtual void InfoPost();          // dump information after finishing
a training epoch
public:
  TrainerNgram(Mach*, ErrFct*, char*, char* =NULL,    // mach, errfct,
train, dev
          float = 0.01, float =0, float =0,     // lrate_beg,
lrate_mult, wdecay
        int =10, int =0);                 // max epochs, current epoch
  TrainerNgram(Mach*, ErrFct*, Data&);    // for testing only
  virtual REAL Train();             // train for one epoch
  virtual REAL TestDev(char * =NULL);     // test current network on
dev data
                              // returns obtained error (-1 if error)
  virtual void TrainAndTest();            // main training routine for
X iterations
};

#endif
```

C. TEGRA K1 DEVICE PROFILE

```
CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GK20A"
  CUDA Driver Version / Runtime Version          6.5 / 6.5
  CUDA Capability Major/Minor version number:    3.2
  Total amount of global memory:                 1894 MBytes (1986494464 bytes)
  ( 1) Multiprocessors, (192) CUDA Cores/MP:     192 CUDA Cores
  GPU Clock rate:                                852 MHz (0.85 GHz)
  Memory Clock rate:                             924 Mhz
  Memory Bus Width:                              64-bit
  L2 Cache Size:                                 131072 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096,
4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            Yes
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           0 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 6.5, CUDA Runtime Version =
6.5, NumDevs = 1, Device0 = GK20A
Result = PASS
```

D. TEGRA K1 GPU IMPLEMENTATION SOURCE FILES

D.1 Original CSLM Version 3.0 GPU.cu

```
/*
 * This file is part of the continuous space language and translation
model toolkit
 * for statistical machine translation and large vocabulary speech
recognition.
 *
 * Copyright 2014, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU Lesser General Public License version 3
as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
License
 * for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
 *
 * $Id: Gpu.cu,v 1.43 2014/03/25 21:52:53 schwenk Exp $
 */

using namespace std;
#include <algorithm>
#include <signal.h>
#define RAISE raise(SIGINT);

typedef float REAL;
#define NULL_WORD (-1)

#include <npps.h>
#include <cublas.h>
#include <cuda_runtime_api.h>
#include "Tools.h" //For Error(
```

```
static REAL *gpu_result;


//-----------------------------------------------
// forward pass for MachTab
//-----------------------------------------------

__global__
void KernelMachTabForw(const int odim, REAL *gpu_data_in, REAL *gpu_t,
REAL *gpu_data_out)
{
  int b=blockIdx.x;
  int i=threadIdx.x;

  int idx= (int) gpu_data_in[b];
  int offso=b*odim;
  int offst=idx*odim;

  if (idx==NULL_WORD) gpu_data_out[i+offso] = 0.0;
                 else gpu_data_out[i+offso] = gpu_t[i+offst];
}

void GpuMachTabForw(const int bsize, const int odim,
                REAL *gpu_data_in, REAL *gpu_t, REAL *gpu_data_out)
{
  KernelMachTabForw<<<bsize,odim>>>(odim, gpu_data_in, gpu_t,
gpu_data_out);
}


//-----------------------------------------------
// backward pass for MachTab
//-----------------------------------------------

__global__
void KernelMachTabBackw(const REAL lrate, const int odim, REAL
*gpu_data_in, REAL *gpu_t, REAL *gpu_grad_out)
{
  int b=blockIdx.x;
  int i=threadIdx.x;

  int idx= (int) gpu_data_in[b];
  if (idx!=NULL_WORD) gpu_t[i+idx*odim] += lrate *
gpu_grad_out[i+b*odim];
}


void GpuMachTabBackw(const REAL lrate, const int bsize, const int odim,
                REAL *gpu_data_in, REAL *gpu_t, REAL *gpu_grad_out)
{
  KernelMachTabBackw<<<bsize,odim>>>(lrate, odim, gpu_data_in, gpu_t,
gpu_grad_out);
}


//-----------------------------------------------
// Softmax normalization
```

```
//------------------------------------------------

//We suppose that x is already the exponant.
//Otherwise, we would compute it twice.
__global__ void KernelSoftmax(int M, int N,
                        const REAL * x, const int sx0, const int sx1,
                        REAL * sm, const int sm_s0, const int sm_s1)
{
  extern __shared__ REAL buf[];
  for (int blockIDX = blockIdx.x; blockIDX < M; blockIDX += gridDim.x)
{
    REAL sum = 0;
#pragma unroll 16
    for (int i = threadIdx.x; i< N; i += blockDim.x){
      sum += exp(x[blockIDX * sx0 + i * sx1]);
    }
    buf[threadIdx.x] = sum;
    __syncthreads();

    // This function trashes buf[1..warpsize], leaving the reduction
result in buf[0].
    if (threadIdx.x < warpSize){
#pragma unroll 8
      for (int i = threadIdx.x + warpSize; i < blockDim.x; i +=
warpSize){
                buf[threadIdx.x] += buf[i];
      }
      if (threadIdx.x < 16){
                //reduce so that threadIdx.x 0 has the sum of
everything
                if(threadIdx.x + 16 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+16];
                if(threadIdx.x + 8 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+8];
                if(threadIdx.x + 4 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+4];
                if(threadIdx.x + 2 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+2];
                if(threadIdx.x + 1 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+1];
      }
    }
    __syncthreads();
    REAL row_sum = buf[0];
#pragma unroll 16
    for (int i = threadIdx.x; i< N; i += blockDim.x){
      sm[blockIDX * sm_s0 + i * sm_s1] = exp(x[blockIDX * sx0 + i *
sx1]) / row_sum;
    }
    __syncthreads();
  }
}
```

```
void GpuMachSoftmaxForw(const int bsize, const int odim, REAL
*gpu_data_out)
{
  if(0){
    //This is the original code that is know to work correctly in all
case,
    //But is slower.
    nppsExp_32f_I(gpu_data_out, bsize*odim);

    REAL sum, *optr=gpu_data_out;

    for (int b=0; b<bsize; b++,optr+=odim) {
      sum=cublasSasum(odim,optr,1);  // exp(x) is always positive -> we
can use the sum_i (ABS(x_i))
      nppsMulC_32f_I(1.0/sum,optr,odim);
    }
    return;
  }

  int warpSize = 32;
  if(odim < warpSize){
    Error("GpuMachSoftmaxForw need an odim >= warpSize(K20 and previous
GPU have a warpSize of 32)");
  }
//The follwing check need to access the GPU properties to do it.
//To don't do this access each time, we have done it in MachSoftmax.cpp
//  if(warpSize != 32){
//    Error("GpuMachSoftmaxForw suppose the warpSize is 32. If run with
a GPU with other warpSize"
//      " like the current GPU, it will return wrong Results. You must
update the reduction in KernelSoftmax");
//  }
  int n_blocks = std::min(bsize, 32 * 1024);
  //TODO, detect the maximum number of thread per block.
  int n_threads = std::min(odim, 512);
  int n_shared_bytes = n_threads * sizeof(REAL);
  if (bsize > 0){
    KernelSoftmax<<<n_blocks, n_threads, n_shared_bytes>>>(
                            bsize,
                            odim,
                            gpu_data_out,
                            odim, //x.stride[0
                            1, //x.stride[1]
                            gpu_data_out,
                            odim, //sm.stride[0]
                            1//sm.stride[1]
                    );
    cudaError_t err = cudaGetLastError();
    if(cudaSuccess != err){
      printf("KernelSoftmax: n_blockn=%d, n_threads=%d,
n_shared_bytes=%d odim=%d\n",
             n_blocks, n_threads, n_shared_bytes, odim);
      Error(cudaGetErrorString(err));
    }
  }
}
```

```
//--------------------------------------------
// Softmax stable normalization
//--------------------------------------------

__global__ void KernelSoftmaxStable(int M, int N,
                                    const REAL * x, const int sx0,
const int sx1,
                                    REAL * sm, const int sm_s0, const
int sm_s1)
{
  extern __shared__ REAL buf[];
  for (int blockIDX = blockIdx.x; blockIDX < M; blockIDX += gridDim.x)
{
    REAL max_ = x[blockIDX * sx0 + threadIdx.x * sx1];
    for (int i = threadIdx.x + blockDim.x; i< N; i += blockDim.x) {
      max_ = max(max_, x[blockIDX * sx0 + i * sx1]);
    };
    buf[threadIdx.x] = max_;
    __syncthreads();

    // This function trashes buf[1..n_threads], leaving the reduction
result in buf[0].
    // Find the max to stabilize the softmax
    if (threadIdx.x < warpSize)
    {
      for (int i = threadIdx.x + warpSize; i < blockDim.x; i +=
warpSize) {
                buf[threadIdx.x] = max(buf[threadIdx.x], buf[i]);
      }
      if (threadIdx.x < 16) {
                //reduce so that threadIdx.x 0 has the max of
everything
                if(threadIdx.x + 16 < N)
                    buf[threadIdx.x] = max(buf[threadIdx.x],
buf[threadIdx.x+16]);
                if(threadIdx.x + 8 < N)
                    buf[threadIdx.x] = max(buf[threadIdx.x],
buf[threadIdx.x+8]);
                if(threadIdx.x + 4 < N)
                    buf[threadIdx.x] = max(buf[threadIdx.x],
buf[threadIdx.x+4]);
                if(threadIdx.x + 2 < N)
                    buf[threadIdx.x] = max(buf[threadIdx.x],
buf[threadIdx.x+2]);
                if(threadIdx.x + 1 < N)
                    buf[threadIdx.x] = max(buf[threadIdx.x],
buf[threadIdx.x+1]);
            }
    }

    __syncthreads();
    REAL row_max = buf[0];
    __syncthreads();
    REAL sum = 0;
    for(int i=threadIdx.x; i<N; i+=blockDim.x){
      sum += exp(x[blockIDX * sx0 + i * sx1] - row_max);
    };
```

```
    buf[threadIdx.x] = sum;
    __syncthreads();

    // This function trashes buf[1..N], leaving the reduction result in
buf[0].
    if (threadIdx.x < warpSize){
      for (int i = threadIdx.x + warpSize; i < blockDim.x; i +=
warpSize){
                buf[threadIdx.x] += buf[i];
      }
      if (threadIdx.x < 16){
                //reduce so that threadIdx.x 0 has the sum of
everything
                if(threadIdx.x + 16 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+16];
                if(threadIdx.x + 8 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+8];
                if(threadIdx.x + 4 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+4];
                if(threadIdx.x + 2 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+2];
                if(threadIdx.x + 1 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+1];
      }
    }
    __syncthreads();
    REAL row_sum = buf[0];
    for (int i = threadIdx.x; i< N; i += blockDim.x){
      sm[blockIDX * sm_s0 + i * sm_s1] = exp(x[blockIDX * sx0 + i *
sx1] - row_max) / row_sum;
    }
    __syncthreads();
  }
}

void GpuMachSoftmaxStableForw(const int bsize, const int odim, REAL
*gpu_data_out)
{
  if(0){
    Error("Not implemented!");
    //This is the original code that is know to work correctly in all
case,
    //But is slower.
    nppsExp_32f_I(gpu_data_out, bsize*odim);

    REAL sum, *optr=gpu_data_out;

    for (int b=0; b<bsize; b++,optr+=odim) {
      sum=cublasSasum(odim,optr,1);  // exp(x) is always positive -> we
can use the sum_i (ABS(x_i))
      nppsMulC_32f_I(1.0/sum,optr,odim);
    }
```

```
    return;
  }
  int warpSize = 32;
  if(odim < warpSize){
    Error("GpuMachSoftmaxStableForw need an odim >= warpSize(K20 and
previous GPU have a warpSize of 32)");
  }
//The follwing check need to access the GPU properties to do it.
//To don't do this access each time, we have done it in
MachSoftmaxStable.cpp
//  if(warpSize != 32){
//    Error("GpuMachSoftmaxStableForw suppose the warpSize is 32. If
run with a GPU with other warpSize"
//        " like the current GPU, it will return wrong Results. You
must update the reduction in KernelSoftmaxStable");
//  }
  int n_blocks = std::min(bsize, 32 * 1024);
  //TODO, detect the maximum number of thread per block.
  int n_threads = std::min(odim, 512);
  int n_shared_bytes = n_threads * sizeof(REAL);
  if (bsize > 0){
    KernelSoftmaxStable<<<n_blocks, n_threads, n_shared_bytes>>>(
                            bsize,
                            odim,
                            gpu_data_out,
                            odim, //x.stride[0]
                            1, //x.stride[1]
                            gpu_data_out,
                            odim, //sm.stride[0]
                            1//sm.stride[1]
                    );
    cudaError_t err = cudaGetLastError();
    if(cudaSuccess != err){
      printf("n_blocks=%d, n_threads=%d, n_shared_bytes=%d odim=%d\n",
             n_blocks, n_threads, n_shared_bytes, odim);
      Error(cudaGetErrorString(err));
    }
  }
}

//----------------------------------------------
// Linear Rectifier units
//----------------------------------------------

__global__
void KernelLinRectifForw(const int n, REAL *gpu_data_out)
{
  int tx = threadIdx.x;
  int bx = blockIdx.x;
  int n_threads = blockDim.x * gridDim.x;
  int id = tx * blockDim.x + bx * gridDim.x;
  for(int i = id; i < n; i += n_threads){
    if (gpu_data_out[i]<0) gpu_data_out[i]=0;
  }
}

void GpuLinRectifForw(const int n, REAL *gpu_data_out)
```

```
{
  int nb_thread = std::min(n, 256);
  int nb_block = n / 256;
  KernelLinRectifForw<<<nb_block, nb_thread>>>(n, gpu_data_out);
}

__global__
void KernelLinRectifBackw(const int n, REAL *gpu_data_out, REAL
*gpu_grad_out)
{
  int tx = threadIdx.x;
  int bx = blockIdx.x;
  int n_threads = blockDim.x * gridDim.x;
  int id = tx * blockDim.x + bx * gridDim.x;
  for(int i = id; i < n; i += n_threads){
    if (gpu_data_out[i]<0) gpu_grad_out[i]=0; else gpu_grad_out[i]=1;
  }
}

void GpuLinRectifBackw(const int n, REAL *gpu_data_out, REAL
*gpu_grad_out)
{
  int nb_thread = std::min(n, 256);
  int nb_block = n / 256;
  KernelLinRectifBackw<<<nb_block, nb_thread>>>(n, gpu_data_out,
gpu_grad_out);
}

//-----------------------------------------------
// Helper functions for drop-out
//-----------------------------------------------

__global__
void KernelDropOut(const int n, REAL *gpu_vect, REAL *rand, REAL
thresh)
{
  int tx = threadIdx.x;
  int bx = blockIdx.x;
  int n_threads = blockDim.x * gridDim.x;
  int id = tx * blockDim.x + bx * gridDim.x;
  REAL coeff=1.0/(1.0-thresh);
  for (int i = id; i < n; i += n_threads) {
    if (rand[i]<thresh) gpu_vect[i]=0.0;
                  else gpu_vect[i]*=coeff;
  }
}

void GpuDropOut(const int n, REAL *gpu_vect, REAL *rand, REAL thresh)
{
  int nb_thread = std::min(n, 256);
  int nb_block = n / 256;
  KernelDropOut<<<nb_block, nb_thread>>>(n, gpu_vect, rand, thresh);
}

//-----------------------------------------------
// ErrFctSoftmCrossEntNgram::CalcValue
//-----------------------------------------------
```

```
__global__
void KernelErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_target,
                                    REAL *gpu_res)
{
  float err=0.0f;
  REAL *optr=gpu_data_out;
  REAL *tptr=gpu_target;
  for (int b=0; b<bsize; b++) {
     err += log(optr[(uint) *tptr++]);
     optr += odim;
  }
  *gpu_res=err;
}


REAL GpuErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_target)
{
  REAL res;
  if (gpu_result==NULL) cudaMalloc(&gpu_result,sizeof(REAL));
  KernelErrFctSoftmCrossEntNgramCalcValue<<<1,1>>>(bsize, odim,
gpu_data_out, gpu_target, gpu_result);
  cudaMemcpy(&res, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);
  return res;
}

//-------------------------------------------------
// ErrFctSoftmCrossEntNgram::CalcGrad
//-------------------------------------------------

__global__
void KernelErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target,
                                    REAL *gpu_res)
{
  extern __shared__ REAL buf[];
  REAL err=0.0;
  for (int b=threadIdx.x; b<bsize; b+=blockDim.x) {
    unsigned int tidx=(uint) gpu_target[b];
    gpu_grad[b*odim + tidx] += 1.0f;
    err += log(gpu_data_out[b*odim + tidx]);
  }
  buf[threadIdx.x] = err;
  __syncthreads();
  if(threadIdx.x == 0){
    for(int i=1; i<blockDim.x;i++){
      err += buf[i];
    }
    *gpu_res=err;
  }
}

void GpuErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL *gpu_data_out,
```

```
                                          REAL *gpu_grad, REAL
*gpu_target, REAL * gpu_res)
{
  cudaMemcpyAsync(gpu_grad, gpu_data_out, bsize*odim*sizeof(REAL),
cudaMemcpyDeviceToDevice);
  nppsMulC_32f_I(-1.0f,gpu_grad,bsize*odim);

  int nb_threads = std::min(512, bsize);
  int n_shared_bytes = nb_threads * sizeof(REAL);
  KernelErrFctSoftmCrossEntNgramCalcGrad<<<1, nb_threads,
n_shared_bytes>>>(
      bsize, odim, gpu_data_out, gpu_grad, gpu_target, gpu_res);

  cudaError_t err = cudaGetLastError();
  if(cudaSuccess != err){
    ErrorN("Error in GpuErrFctSoftmCrossEntNgramCalcGrad: %s",
cudaGetErrorString(err));
  }
}

//-----------------------------------------------
// ErrFctSoftmCrossEntNgram::CalcGradNull
//-----------------------------------------------
//This kernel is special, we need many block to do the memset of
NULL_WORD
//But also need to do a reduction.
//So the first block will do the reduction and the update of the grad
associated with it
//But all the other will only be there to memset the grad of NULL_WORD.
__global__
void KernelErrFctSoftmCrossEntNgramCalcGradNull(const int bsize, const
int odim,
     REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target,
                                          REAL *gpu_res)
{
  extern __shared__ REAL buf[];
  REAL err=0.0;
if(blockIdx.x == 0){
//The first block compute the errors and grad for non NULL work
  for (int b=threadIdx.x; b<bsize; b+=blockDim.x) {
    //Do not cast or use unsigned for tidx. Otherwise, nvcc will
transform the -1 to 0!
    //This is a difference compared to the GPU!
    int tidx = gpu_target[b];
    if (tidx==NULL_WORD) {
      //memset(&gpu_grad[b*odim], 0, odim*sizeof(REAL));
    } else {
      gpu_grad[b*odim + tidx] += 1.0f;
      err += log(gpu_data_out[b*odim + tidx]);
    }
  }
  buf[threadIdx.x] = err;
  __syncthreads();
  if(threadIdx.x == 0){
    for(int i=1; i<blockDim.x;i++){
      err += buf[i];
    }
```

```
    *gpu_res=err;
  }
}else{
//All the other block computer just the grad for NULL word
  for(int b=blockIdx.x-1;b<bsize;b+=gridDim.x-1){
    int tidx = gpu_target[b];
    if (tidx==NULL_WORD) {
      for (int i=threadIdx.x; i<odim; i+=blockDim.x)
        gpu_grad[b*odim + i] = 0;
    }

  }
}
}

void GpuErrFctSoftmCrossEntNgramCalcGradNull(const int bsize, const int
odim, REAL *gpu_data_out,
                                             REAL *gpu_grad, REAL
*gpu_target, REAL * gpu_res)
{
  cudaMemcpyAsync(gpu_grad, gpu_data_out, bsize*odim*sizeof(REAL),
cudaMemcpyDeviceToDevice);
  nppsMulC_32f_I(-1.0f,gpu_grad,bsize*odim);

  int nb_threads = std::min(512, bsize);
  int n_shared_bytes = nb_threads * sizeof(REAL);
  KernelErrFctSoftmCrossEntNgramCalcGradNull<<<bsize+1, nb_threads,
n_shared_bytes>>>(
      bsize, odim, gpu_data_out, gpu_grad, gpu_target, gpu_res);

  cudaError_t err = cudaGetLastError();
  if(cudaSuccess != err){
    ErrorN("Error in GpuErrFctSoftmCrossEntNgramCalcGradNull: %s",
cudaGetErrorString(err));
  }
}

//-------------------------------------------------
// ErrFctSoftmCrossEntNgram::CalcGradCumul
//-------------------------------------------------

__global__
void KernelErrFctSoftmCrossEntNgramCalcGradCumul(const int bsize, const
int odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target,
                                    REAL *gpu_res)
{
  REAL *optr=gpu_data_out;
  REAL *gptr=gpu_grad;
  REAL *tptr=gpu_target;
  REAL err=0.0;
  unsigned int tidx;

 for (int b=0; b<bsize; b++) {
    tidx=(uint) *tptr++;
    gptr[tidx] += 1.0f;
    err += log(optr[tidx]);  // modify to run log in parallel
    gptr+=odim; optr+=odim;
```

```
  }
  *gpu_res+=err;
}


void GpuErrFctSoftmCrossEntNgramCalcGradCumul(const int bsize, const
int odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target)
{
  if (gpu_result==NULL) cudaMalloc(&gpu_result,sizeof(REAL));

  cudaMemcpy(gpu_grad, gpu_data_out, bsize*odim*sizeof(REAL),
cudaMemcpyDeviceToDevice);
  nppsMulC_32f_I(-1.0f,gpu_grad,bsize*odim);
  KernelErrFctSoftmCrossEntNgramCalcGradCumul<<<1,1>>>(bsize, odim,
gpu_data_out, gpu_grad, gpu_target, gpu_result);
  Error("GpuErrFctSoftmCrossEntNgramCalcGradCumul not finished!");

  //REAL res;
  //cudaMemcpy(&res, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);
  //return res;
}

//-------------------------------------------------
// ErrFctSoftmCrossEntNgramMulit::CalcGrad
//-------------------------------------------------

__global__
void KernelErrFctSoftmCrossEntNgramMultiCalcGrad(const int bsize, const
int dim, const int nb,
                                        REAL *gpu_data_out, REAL
*gpu_grad, REAL *gpu_target,
                                                  REAL *gpu_res)
{
  extern __shared__ REAL buf[];
  REAL err=0.0;
  for (int b=blockIdx.x; b<bsize; b+=gridDim.x) {
    for (int n=threadIdx.x; n<nb; n+=blockDim.x) {
      int tidx=(int) gpu_target[b*nb + n];// *tptr++;
      if (tidx==NULL_WORD) {
        //memset(gptr, 0, dim*sizeof(REAL));
        for(int i=0;i<dim;i++)
            gpu_grad[(b*nb+n)*dim + i] = 0;
      }
      else {
        gpu_grad[(b*nb+n)*dim + tidx] += 1.0;
        err += log(gpu_data_out[(b*nb+n)*dim + tidx]);
      }
    }
  }
  buf[threadIdx.x] = err;
  __syncthreads();
  if(threadIdx.x == 0){
    for(int i=1; i<blockDim.x;i++){
      err += buf[i];
    }
    atomicAdd(gpu_res, err);
  }
```

```
}

REAL GpuErrFctSoftmCrossEntNgramMultiCalcGrad(const int bsize, const
int dim, const int nb,
                                                REAL *gpu_data_out, REAL
*gpu_grad, REAL *gpu_target)
{
  if (gpu_result==NULL) cudaMalloc(&gpu_result, sizeof(REAL));

// same below
  int n=bsize*nb*dim;
  cudaMemcpy(gpu_grad, gpu_data_out, n*sizeof(REAL),
             cudaMemcpyDeviceToDevice);

  nppsMulC_32f_I(-1.0f,gpu_grad,bsize*nb*dim);
  cudaMemset(gpu_result, 0.0, sizeof(REAL));//Each block will atomicAdd
into it.

  cudaError_t sts = cudaGetLastError();
  if (cudaSuccess != sts)
    Error("Error before KernelErrFctSoftmCrossEntNgramMultiCalcGrad");
  int nb_threads = std::min(nb, 1024);
  int nb_blocks = std::min(bsize, 1 << 16);

KernelErrFctSoftmCrossEntNgramMultiCalcGrad<<<nb_blocks,nb_threads,nb_t
hreads*sizeof(REAL)>>>(
    bsize, dim, nb, gpu_data_out, gpu_grad, gpu_target, gpu_result);
  sts = cudaGetLastError();
  if (cudaSuccess != sts)
  {
    printf(cudaGetErrorString(sts));
    Error("KernelErrFctSoftmCrossEntNgramMultiCalcGrad cuda error: ");
  }
  REAL res;
  cudaMemcpy(&res, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);

  return res;
}

//-------------------------------------------------
// Copy
//-------------------------------------------------
__global__
void KernelCopyVectorToMatrix(REAL * mat, REAL * vec, const int M,
const int N)
{
  for(int b = blockIdx.x; b<M; b+=gridDim.x)
    for(int i = threadIdx.x; i<N; i+=blockDim.x)
      mat[b * N + i] = vec[i];
}

/*
 * This copy the vector on each line of the matrix.
 */
void GpuCopyVectorToMatrix(REAL * mat, REAL * vec, const int M, const
int N)
{
```

```
  int nb_blocks = std::min(M, 1 << 16);
  int nb_threads = std::min(N, 512);
  KernelCopyVectorToMatrix<<<nb_blocks, nb_threads>>>(mat, vec, M, N);
  cudaError_t cuda_stat=cudaGetLastError();
  if (cuda_stat != cudaSuccess)
  { ErrorN("CUDA: ERROR %d in GpuCopyVectorToMatrix(%p, %p %d %d):
%s\n",
           cuda_stat, mat, vec, M, N, cudaGetErrorString(cuda_stat));
  }
}

__global__
void KernelCopyMatrixToMatrixStrided(REAL * dst, REAL * src, const int
M, const int N, const int row_stride)
{
  for(int b = blockIdx.x; b<M; b+=gridDim.x)
    for(int i = threadIdx.x; i<N; i+=blockDim.x)
      dst[b * row_stride + i] = src[b * N + i];
}

__global__
void KernelCopyMatrixStridedToMatrix(REAL * dst, REAL * src, const int
M, const int N,
                                     const int row_stride_src)
{
  for(int b = blockIdx.x; b<M; b+=gridDim.x)
    for(int i = threadIdx.x; i<N; i+=blockDim.x)
      dst[b * N + i] = src[b * row_stride_src + i];
}

/*
 * This copy each line of a contiguous matrix to another matrix that is
strided
 */
void GpuCopyMatrixToMatrixStrided(REAL * dst, REAL * src, const int M,
const int N, const int row_stride)
{
  int nb_blocks = std::min(M, 1 << 16);
  int nb_threads = std::min(N, 512);
  KernelCopyMatrixToMatrixStrided<<<nb_blocks, nb_threads>>>(dst, src,
M, N, row_stride);
  cudaError_t cuda_stat=cudaGetLastError();
  if (cuda_stat != cudaSuccess){
    ErrorN("CUDA: ERROR %d in GpuCopyMatrixToMatrixStrided: %s\n",
           cuda_stat, cudaGetErrorString(cuda_stat));
  }
}

/*
 * This copy each line of a strided matrix to another matrix that is
contiguous
 */
void GpuCopyMatrixStridedToMatrix(REAL * dst, REAL * src, const int M,
const int N, const int row_stride)
{
  int nb_blocks = std::min(M, 1 << 16);
  int nb_threads = std::min(N, 512);
```

```
  KernelCopyMatrixStridedToMatrix<<<nb_blocks, nb_threads>>>(dst, src,
M, N, row_stride);
  cudaError_t cuda_stat=cudaGetLastError();
  if (cuda_stat != cudaSuccess){
    ErrorN("CUDA: ERROR %d in GpuCopyMatrixToMatrixStrided: %s\n",
           cuda_stat, cudaGetErrorString(cuda_stat));
  }
}

//------------------------------------------------
// Multiple AXPY input row on one output row
//------------------------------------------------

// Each block compute a fixed number of colums for all batch.
// This allow to have read coalesced and don't need atomic opartion.
__global__
void KernelBatchedAXPY(const int n, const REAL a, REAL * x, const int
incx,
                       REAL * y, const int incy, const int nb_batch){
  for(int idx = blockIdx.x * blockDim.x + threadIdx.x; idx < n;
      idx += blockDim.x*gridDim.x){
    for(int b=0; b<nb_batch; b++){
      y[idx * incy] += a * x[b * n * incx + idx * incx];
    }
  }
}

void GpuBatchedAXPY(const int n, const REAL a, REAL * x, const int
incx,
                    REAL * y, const int incy, const int nb_batch){
  int nb_threads = std::min(128, n);
  int nb_blocks = std::min(1<<16, n/nb_threads+(n%nb_threads==0?0:1));
  nb_blocks = std::max(nb_blocks, 1);
  KernelBatchedAXPY<<<nb_blocks,nb_threads>>>(n, a, x, incx, y, incy,
nb_batch);

}

//------------------------------------------------
// Helpers
//------------------------------------------------


void GpuResSet(REAL val) {
  cudaMemcpy(gpu_result, &val, sizeof(REAL), cudaMemcpyHostToDevice);
}

REAL GpuResGet() {
  REAL val;
  cudaMemcpy(&val, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);
  return val;
}
```

D.2 Original CSLM Version 2.0 GPU.cu

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2012, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Gpu.cu,v 1.10 2012/06/03 22:14:18 schwenk Exp $
 */

using namespace std;

typedef float REAL;
#define NULL_WORD (-1)

#include <npps.h>
#include <cublas.h>
#include <cuda_runtime_api.h>

static REAL *gpu_result;


//-----------------------------------------------
// forward pass for MachTab
//-----------------------------------------------

__global__
void KernelMachTabForw(const int odim, REAL *gpu_data_in, REAL *gpu_t,
REAL *gpu_data_out)
{
  int b=blockIdx.x;
  int i=threadIdx.x;

  int idx= (int) gpu_data_in[b];
  int offso=b*odim;
  int offst=idx*odim;
```

```
  if (idx==NULL_WORD) gpu_data_out[i+offso] = 0.0;
                 else gpu_data_out[i+offso] = gpu_t[i+offst];
}

void GpuMachTabForw(const int bsize, const int odim,
                REAL *gpu_data_in, REAL *gpu_t, REAL *gpu_data_out)
{
  KernelMachTabForw<<<bsize,odim>>>(odim, gpu_data_in, gpu_t,
gpu_data_out);
}


//-----------------------------------------------
// backward pass for MachTab
//-----------------------------------------------

__global__
void KernelMachTabBackw(const REAL lrate, const int odim, REAL
*gpu_data_in, REAL *gpu_t, REAL *gpu_grad_out)
{
  int b=blockIdx.x;
  int i=threadIdx.x;

  int idx= (int) gpu_data_in[b];
  if (idx!=NULL_WORD) gpu_t[i+idx*odim] += lrate *
gpu_grad_out[i+b*odim];
}


void GpuMachTabBackw(const REAL lrate, const int bsize, const int odim,
                REAL *gpu_data_in, REAL *gpu_t, REAL *gpu_grad_out)
{
  KernelMachTabBackw<<<bsize,odim>>>(lrate, odim, gpu_data_in, gpu_t,
gpu_grad_out);
}


//-----------------------------------------------
// Softmax normalization
//-----------------------------------------------

void GpuMachSoftmaxForw(const int bsize, const int odim, REAL
*gpu_data_out)
{
  nppsExp_32f_I(gpu_data_out, bsize*odim);

  REAL sum, *optr=gpu_data_out;

  for (int b=0; b<bsize; b++,optr+=odim) {
    sum=cublasSasum(odim,optr,1);  // exp(x) is always positive -> we
can use the sum_i (ABS(x_i))
    nppsMulC_32f_I(1.0/sum,optr,odim);
  }
}

//-----------------------------------------------
// ErrFctSoftmCrossEntNgram::CalcValue
```

```
//------------------------------------------------

__global__
void KernelErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_target,
                                     REAL *gpu_res)
{
  float err=0.0f;
  REAL *optr=gpu_data_out;
  REAL *tptr=gpu_target;
  for (int b=0; b<bsize; b++) {
     err += log(optr[(uint) *tptr++]);
     optr += odim;
  }
  *gpu_res=err;
}


REAL GpuErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_target)
{
  REAL res;
  if (gpu_result==NULL) cudaMalloc(&gpu_result,sizeof(REAL));
  KernelErrFctSoftmCrossEntNgramCalcValue<<<1,1>>>(bsize, odim,
gpu_data_out, gpu_target, gpu_result);
  cudaMemcpy(&res, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);
  return res;
}

//------------------------------------------------
// ErrFctSoftmCrossEntNgram::CalcGrad
//------------------------------------------------

__global__
void KernelErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target,
                                    REAL *gpu_res)
{
  REAL *optr=gpu_data_out;
  REAL *gptr=gpu_grad;
  REAL *tptr=gpu_target;
  REAL err=0.0;
  unsigned int tidx;

 for (int b=0; b<bsize; b++) {
    tidx=(uint) *tptr++;
    gptr[tidx] += 1.0f;
    err += log(optr[tidx]);  // modify to run log in parallel
    gptr+=odim; optr+=odim;
  }
  *gpu_res=err;
}

__global__
void KernelErrFctSoftmCrossEntNgramCalcGradCumul(const int bsize, const
int odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target,
                                    REAL *gpu_res)
```

```
{
  REAL *optr=gpu_data_out;
  REAL *gptr=gpu_grad;
  REAL *tptr=gpu_target;
  REAL err=0.0;
  unsigned int tidx;

 for (int b=0; b<bsize; b++) {
    tidx=(uint) *tptr++;
    gptr[tidx] += 1.0f;
    err += log(optr[tidx]);  // modify to run log in parallel
    gptr+=odim; optr+=odim;
  }
  *gpu_res+=err;
}


REAL GpuErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target)
{
  if (gpu_result==NULL) cudaMalloc(&gpu_result,sizeof(REAL));

  cudaMemcpy(gpu_grad, gpu_data_out, bsize*odim*sizeof(REAL),
cudaMemcpyDeviceToDevice);
  nppsMulC_32f_I(-1.0f,gpu_grad,bsize*odim);

  REAL res;
  KernelErrFctSoftmCrossEntNgramCalcGrad<<<1,1>>>(bsize, odim,
gpu_data_out, gpu_grad, gpu_target, gpu_result);
  cudaMemcpy(&res, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);
  return res;
}

void GpuErrFctSoftmCrossEntNgramCalcGradCumul(const int bsize, const
int odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target)
{
  if (gpu_result==NULL) cudaMalloc(&gpu_result,sizeof(REAL));

  cudaMemcpy(gpu_grad, gpu_data_out, bsize*odim*sizeof(REAL),
cudaMemcpyDeviceToDevice);
  nppsMulC_32f_I(-1.0f,gpu_grad,bsize*odim);
  KernelErrFctSoftmCrossEntNgramCalcGradCumul<<<1,1>>>(bsize, odim,
gpu_data_out, gpu_grad, gpu_target, gpu_result);
}


void GpuResSet(REAL val) {
  cudaMemcpy(gpu_result, &val, sizeof(REAL), cudaMemcpyHostToDevice);
}

REAL GpuResGet() {
  REAL val;
  cudaMemcpy(&val, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);
  return val;
}
```

D.3 Tegra K1 GPU Implementation GPU.cu

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2012, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Gpu.cu,v 1.10 2012/06/03 22:14:18 schwenk Exp $
 */

using namespace std;
#include <iostream>

typedef float REAL;
#define NULL_WORD (-1)

#include "Tools.h"
#include <npps.h>
#include <cublas.h>
#include <cuda_runtime_api.h>

static REAL *gpu_result;


//-------------------------------------------------
// forward pass for MachTab
//-------------------------------------------------
//From Version 2
__global__
void KernelMachTabForw(const int odim, REAL * gpu_data_in, REAL *
gpu_t, REAL * gpu_data_out)
{
  int b=blockIdx.x;
  int i=threadIdx.x;

  int idx= (int) gpu_data_in[b];
  int offso=b*odim;
```

```
  int offst=idx*odim;

  if (idx==NULL_WORD) gpu_data_out[i+offso] = 0.0;
                 else gpu_data_out[i+offso] = gpu_t[i+offst];
}

//From Version 2
void GpuMachTabForw(const int bsize, const int odim, REAL *
gpu_data_in, REAL * gpu_t, REAL * gpu_data_out)
{
  KernelMachTabForw<<<bsize,odim>>>(odim, gpu_data_in, gpu_t,
gpu_data_out);
}

//From Version 2
//-----------------------------------------------
// backward pass for MachTab
//-----------------------------------------------

__global__
void KernelMachTabBackw(const REAL lrate, const int odim, REAL *
gpu_data_in, REAL * gpu_t, REAL * gpu_grad_out)
{
  int b=blockIdx.x;
  int i=threadIdx.x;

  int idx= (int) gpu_data_in[b];
  if (idx!=NULL_WORD) gpu_t[i+idx*odim] += lrate *
gpu_grad_out[i+b*odim];
}

//From Version 2
void GpuMachTabBackw(const REAL lrate, const int bsize, const int odim,
                REAL * gpu_data_in, REAL * gpu_t, REAL * gpu_grad_out)
{
  KernelMachTabBackw<<<bsize,odim>>>(lrate, odim, gpu_data_in, gpu_t,
gpu_grad_out);
}

//From Version 3.0
//-----------------------------------------------
// Softmax normalization
//-----------------------------------------------

//We suppose that x is already the exponant.
//Otherwise, we would compute it twice.
__global__ void KernelSoftmax(int M, int N, const REAL * x, const int
sx0, const int sx1, REAL * sm, const int sm_s0, const int sm_s1)
{
  extern __shared__ REAL buf[];

  for (int blockIDX = blockIdx.x; blockIDX < M; blockIDX += gridDim.x)
{
    REAL sum = 0;
#pragma unroll 16
    for (int i = threadIdx.x; i< N; i += blockDim.x){
      sum += exp(x[blockIDX * sx0 + i * sx1]);
```

```
    }
    buf[threadIdx.x] = sum;
    __syncthreads();

    // This function trashes buf[1..warpsize], leaving the reduction
result in buf[0].
    if (threadIdx.x < warpSize){
#pragma unroll 8
      for (int i = threadIdx.x + warpSize; i < blockDim.x; i +=
warpSize){
                buf[threadIdx.x] += buf[i];
      }
      if (threadIdx.x < 16){
                //reduce so that threadIdx.x 0 has the sum of
everything
                if(threadIdx.x + 16 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+16];
                if(threadIdx.x + 8 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+8];
                if(threadIdx.x + 4 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+4];
                if(threadIdx.x + 2 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+2];
                if(threadIdx.x + 1 < N)
                    buf[threadIdx.x] = buf[threadIdx.x] +
buf[threadIdx.x+1];
      }
    }
    __syncthreads();
    REAL row_sum = buf[0];
#pragma unroll 16
    for (int i = threadIdx.x; i< N; i += blockDim.x){
      sm[blockIDX * sm_s0 + i * sm_s1] = exp(x[blockIDX * sx0 + i *
sx1]) / row_sum;
    }
    __syncthreads();
  }
}

//From Version 3.0
void GpuMachSoftmaxForw(const int bsize, const int odim, REAL
*data_out)
{

//  int warpSize = 32;
//  if(odim < warpSize){
//    Error("GpuMachSoftmaxForw need an odim >= warpSize(K20 and
previous GPU have a warpSize of 32)");
//  }
//The follwing check need to access the GPU properties to do it.
//To don't do this access each time, we have done it in MachSoftmax.cpp
//  if(warpSize != 32){
```

```
//    Error("GpuMachSoftmaxForw suppose the warpSize is 32. If run with
a GPU with other warpSize"
//      " like the current GPU, it will return wrong Results. You must
update the reduction in KernelSoftmax");
//  }
  int n_blocks = std::min(bsize, 32 * 2048);
  //TODO, detect the maximum number of thread per block.
  int n_threads = std::min(odim, 512);
  int n_shared_bytes = n_threads * sizeof(REAL);
  if (bsize > 0){
    KernelSoftmax<<<n_blocks, n_threads, n_shared_bytes>>>(
                            bsize,
                            odim,
                            gpu_data_out,
                            odim, //x.stride[0
                            1, //x.stride[1]
                            gpu_data_out,
                            odim, //sm.stride[0]
                            1//sm.stride[1]
                    );
    cudaError_t err = cudaGetLastError();
    if(cudaSuccess != err){
      printf("KernelSoftmax: n_blockn=%d, n_threads=%d,
n_shared_bytes=%d odim=%d\n", n_blocks, n_threads, n_shared_bytes,
odim);
      printf("cuda softmax kernel error \n");
      //Error(cudaGetErrorString(err));
    }
  }
}

//From Version 2
//-------------------------------------------------
// ErrFctSoftmCrossEntNgram::CalcValue
//-------------------------------------------------

__global__
void KernelErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL *data_out, REAL *target,REAL *gpu_res)
{
  float err=0.0f;
  REAL *optr=data_out;
  REAL *tptr=target;
  for (int b=0; b<bsize; b++) {
     err += log(optr[(uint) *tptr++]);
     optr += odim;
  }
 *gpu_res=err;
}

//From Version 2
REAL GpuErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL * gpu_data_out, REAL * gpu_target)
{
  REAL res;
  cudaMalloc(&gpu_result,sizeof(REAL));
```

```
  KernelErrFctSoftmCrossEntNgramCalcValue<<<1,1>>>(bsize, odim,
gpu_data_out, gpu_target, gpu_result);
  cudaMemcpy(&res, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);
  cudaFree(gpu_result);
  return res;
}

//From Version 2
//-----------------------------------------------
// ErrFctSoftmCrossEntNgram::CalcGrad
//-----------------------------------------------

__global__
void KernelErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL * gpu_data_out, REAL * gpu_grad, REAL * gpu_target,
                                    REAL *gpu_res)
{
  REAL *optr= gpu_data_out;
  REAL *gptr= gpu_grad;
  REAL *tptr= gpu_target;
  REAL err=0.0;
  unsigned int tidx;

 for (int b=0; b<bsize; b++) {
    tidx=(uint) *tptr++;
    gptr[tidx] += 1.0f;
    err += log(optr[tidx]);  // modify to run log in parallel
    gptr+=odim;
    optr+=odim;
  }
  *gpu_res=err;
}

//From Version 2
REAL GpuErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL * gpu_data_out, REAL * gpu_grad, REAL * gpu_target)
{
  cudaMalloc(&gpu_result,sizeof(REAL));

  cudaMemcpy(gpu_grad, gpu_data_out, bsize*odim*sizeof(REAL),
  cudaMemcpyDeviceToDevice);

  nppsMulC_32f_I(-1.0f, gpu_grad,bsize*odim);

  REAL res;
  KernelErrFctSoftmCrossEntNgramCalcGrad<<<1,1>>>(bsize, odim,
  gpu_data_out, gpu_grad, gpu_target, gpu_result);
  cudaMemcpy(&res, gpu_result, sizeof(REAL), cudaMemcpyDeviceToHost);
  cudaFree(gpu_result);
  return res;
}


//From Version 3.0
//-----------------------------------------------
// Multiple AXPY input row on one output row
//-----------------------------------------------
```

```
// Each block compute a fixed number of colums for all batch.
// This allow to have read coalesced and don't need atomic opartion.
__global__
void KernelBatchedAXPY(const int n, const REAL a, REAL * x, const int
incx, REAL * y, const int incy, const int nb_batch){
  for(int idx = blockIdx.x * blockDim.x + threadIdx.x; idx < n;
      idx += blockDim.x*gridDim.x){
    for(int b=0; b<nb_batch; b++){
      y[idx * incy] += a * x[b * n * incx + idx * incx];
    }
  }
}

//From Version 3.0
void GpuBatchedAXPY(const int n, const REAL a, REAL * x, const int
incx, REAL * y, const int incy, const int nb_batch){
  int nb_threads = std::min(128, n);
  int nb_blocks = std::min(1<<16, n/nb_threads+(n%nb_threads==0?0:1));
  nb_blocks = std::max(nb_blocks, 1);
  KernelBatchedAXPY<<<nb_blocks,nb_threads>>>(n, a, x, incx, y, incy,
nb_batch);

}

//-----------------------------------------------
// Copy
//-----------------------------------------------
__global__
void KernelCopyVectorToMatrix(REAL * mat, REAL * vec, const int M,
const int N)
{
  for(int b = blockIdx.x; b<M; b+=gridDim.x)
    for(int i = threadIdx.x; i<N; i+=blockDim.x)
      mat[b * N + i] = vec[i];
}

/*
 * This copy the vector on each line of the matrix.
 */
void GpuCopyVectorToMatrix(REAL * mat, REAL * vec, const int M, const
int N)
{
  int nb_blocks = std::min(M, 1 << 16);
  int nb_threads = std::min(N, 512);
  KernelCopyVectorToMatrix<<<nb_blocks, nb_threads>>>(mat, vec, M, N);
  cudaError_t cuda_stat=cudaGetLastError();
  if (cuda_stat != cudaSuccess)
  { printf("CUDA: ERROR %d in GpuCopyVectorToMatrix(%p, %p %d %d):
%s\n",cuda_stat, mat, vec, M, N, cudaGetErrorString(cuda_stat));
  }
}
```

D.4 Original CSLM Version 3.0 GPU.cuh

```
/*
 * This file is part of the continuous space language and translation
model toolkit
 * for statistical machine translation and large vocabulary speech
recognition.
 *
 * Copyright 2014, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU Lesser General Public License version 3
as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
License
 * for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
 *
 * $Id: Gpu.cuh,v 1.19 2014/03/25 21:52:53 schwenk Exp $
 */


#ifndef _Gpu_cuh
#define _Gpu_cuh

void GpuMachTabForw(const int bsize, const int odim,
                REAL *gpu_data_in, REAL *gpu_t, REAL *gpu_data_out);

void GpuMachTabBackw(const REAL lrate, const int bsize, const int odim,
                REAL *gpu_data_in, REAL *gpu_t, REAL *gpu_grad_out);

void GpuMachSoftmaxForw(const int bsize, const int odim, REAL
*gpu_data_out);
void GpuMachSoftmaxStableForw(const int bsize, const int odim, REAL
*gpu_data_out);

void GpuLinRectifForw(const int n, REAL *gpu_data_out);
void GpuLinRectifBackw(const int n, REAL *gpu_data_out, REAL
*gpu_grad_out);

void GpuDropOut(const int n, REAL *gpu_vect, REAL *rand, REAL thresh);

REAL GpuErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_target);
```

```
void GpuErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target, REAL *
gpu_res);
void GpuErrFctSoftmCrossEntNgramCalcGradNull(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target, REAL *
gpu_res);
void GpuErrFctSoftmCrossEntNgramCalcGradCumul(const int bsize, const
int odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target);

REAL GpuErrFctSoftmCrossEntNgramMultiCalcGrad(const int bsize, const
int dim, const int nb, REAL *gpu_data_out, REAL *gpu_grad, REAL
*gpu_target);

void GpuCopyVectorToMatrix(REAL * mat, REAL * vec, const int M, const
int N);

void GpuCopyMatrixToMatrixStrided(REAL * dst, REAL * src, const int M,
const int N, const int row_stride);

void GpuCopyMatrixStridedToMatrix(REAL * dst, REAL * src, const int M,
const int N, const int row_stride);

void GpuBatchedAXPY(const int n, const REAL a, REAL * x, const int
incx,
                    REAL * y, const int incy, const int nb_batch);

void GpuResSet(REAL val);
REAL GpuResGet();

#endif
```

D.5 Original CSLM Version 2.0 GPU.cuh

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2012, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Gpu.cuh,v 1.6 2012/06/02 13:24:16 schwenk Exp $
 */


void GpuMachTabForw(const int bsize, const int odim,
                REAL *gpu_data_in, REAL *gpu_t, REAL *gpu_data_out);

void GpuMachTabBackw(const REAL lrate, const int bsize, const int odim,
                REAL *gpu_data_in, REAL *gpu_t, REAL *gpu_grad_out);

void GpuMachSoftmaxForw(const int bsize, const int odim, REAL
*gpu_data_out);

REAL GpuErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_target);

REAL GpuErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target);

void GpuErrFctSoftmCrossEntNgramCalcGradCumul(const int bsize, const
int odim, REAL *gpu_data_out, REAL *gpu_grad, REAL *gpu_target);

void GpuResSet(REAL val);
REAL GpuResGet();
```

D.6 Tegra K1 GPU Implementation GPU.cuh

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2012, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
```

```
 * $Id: Gpu.cuh,v 1.6 2012/06/02 13:24:16 schwenk Exp $
 */

//From version 2
void GpuMachTabForw(const int bsize, const int odim, REAL *gpu_data_in,
REAL * gpu_t, REAL * gpu_data_out);

//From version 2
void GpuMachTabBackw(const REAL lrate, const int bsize, const int odim,
REAL * gpu_data_in, REAL * gpu_t, REAL * gpu_grad_out);

//From Version 3.0
void GpuMachSoftmaxForw(const int bsize, const int odim, REAL
*gpu_data_out);

//From version 2
REAL GpuErrFctSoftmCrossEntNgramCalcValue(const int bsize, const int
odim, REAL * gpu_data_out, REAL * gpu_target);

//From version 2
REAL GpuErrFctSoftmCrossEntNgramCalcGrad(const int bsize, const int
odim, REAL * gpu_data_out, REAL * gpu_grad, REAL * gpu_target);

//From Version 3.0
void GpuCopyVectorToMatrix(REAL * mat, REAL * vec, const int M, const
int N);

//From Version 3.0
void GpuBatchedAXPY(const int n, const REAL a, REAL * x, const int
incx, REAL * y, const int incy, const int nb_batch);
```

D.7 Blas.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
```

```
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Blas.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Blas_h
#define _Blas_h

#include <string.h>     // memcpy()
#include "Tools.h"
#include "cblas.h" //openblas header file
//#include "cublas_v2.h" //cublas header file
//#include "cuda_runtime.h" //needed with cublas

// BLAS helper functions

#ifdef BLAS_INTEL_MKL
  #include "mkl_blas.h"
  #include "mkl_vml.h"
// for single precision
  #define VTANH vstanh_
  #define VEXP vsexp_
  #define VLOG vslog_
  #define GEMV sgemv
  #define GEMM sgemm
#endif

#ifdef BLAS_STD
//extern "C" void sgemv_(const char *trans, const int *m, const int *n,
const float *alpha,
//                const float *a, const int *lda, const float *x, const
int *incx,
//                const float *beta, float *y, const int *incy);
extern "C" void cblas_sgemv(OPENBLAS_CONST enum CBLAS_ORDER order,
OPENBLAS_CONST enum CBLAS_TRANSPOSE trans,  OPENBLAS_CONST blasint m,
OPENBLAS_CONST blasint n,
             OPENBLAS_CONST float alpha, OPENBLAS_CONST float  *a,
OPENBLAS_CONST blasint lda,  OPENBLAS_CONST float  *x, OPENBLAS_CONST
blasint incx,  OPENBLAS_CONST float beta,  float  *y, OPENBLAS_CONST
blasint incy);
//extern "C" void sgemm_(const char *transa, const char *transb, const
int *m, const int *n, const int *k,
//                const float *alpha, const float *a, const int *lda,
const float *b, const int *ldb,
//                const float *beta, float *c, const int *ldc);
extern "C" void cblas_sgemm(OPENBLAS_CONST enum CBLAS_ORDER Order,
OPENBLAS_CONST enum CBLAS_TRANSPOSE TransA, OPENBLAS_CONST enum
CBLAS_TRANSPOSE TransB, OPENBLAS_CONST blasint M, OPENBLAS_CONST
blasint N, OPENBLAS_CONST blasint K,
             OPENBLAS_CONST float alpha, OPENBLAS_CONST float *A,
OPENBLAS_CONST blasint lda, OPENBLAS_CONST float *B, OPENBLAS_CONST
blasint ldb, OPENBLAS_CONST float beta, float *C, OPENBLAS_CONST
blasint ldc);

  //#define GEMV sgemv_
  #define GEMV cblas_sgemv
  //#define GEMM sgemm_
```

```
  #define GEMM cblas_sgemm
#endif

// matrix/vector multiplication: c = 1.0*A * b + 1.0 * c
// the matrix must be stored in COLUM MAJOR order

/*--------------------------------------------------------------------
-----*
 *
 * Wrapper routine for GEMV function
 * that uses the TRANSPOSED fortran routine
 *
 * dest = matrix * source + bias
 *
 *   dest: dim_dest x 1
 * matrix: dim_dest x dim_src
 * source: dim_src x 1
 *
 *--------------------------------------------------------------------
-----*/

inline void call_gemv (REAL *dest, REAL *matrix, REAL *source, REAL
*bias,
                       int dim_dest, int dim_src)
{
      //char    trans = 'N';
      REAL    fact = 1.0;
      int         inc = 1;

      // int sgemv(char *trans, integer *m, integer *n,
      //     real *alpha, *real *a, integer *lda,
      //     real *x, integer *incx, real *beta, real *y, *integer
*incy)
      //
      // y := alpha*A*x + beta*y
      //         m x n


      debug("-mkl- call gemv\n");
      memcpy (dest, bias, dim_dest * sizeof(REAL));
      //GEMV (CblasColMajor,CblasNoTrans, &dim_dest, &dim_src, &fact,
matrix, &dim_dest, source, &inc, &fact, dest, &inc);
        GEMV (CblasColMajor,CblasNoTrans, dim_dest, dim_src, fact,
matrix, dim_dest, source, inc, fact, dest, inc);
}


// matrix/matrix multiplication: C = alpha*A * B + beta * b
// both must be stored in COLUM MAJOR order

inline void call_gemm (REAL *C, REAL *A, REAL *B, REAL beta, int dimy,
int dimx, int dimk)
{
  //char    transN = 'N';
//  float    alpha = 1.0f;
//  int n2A = dimy * dimk;
//  int n2B = dimk * dimx;
```

```
//  int n2C = dimy * dimx;
  //int i = 0;
//  REAL *d_A = 0;
//  REAL *d_B = 0;
//  REAL *d_C = 0;
  //REAL *h_A;
  //REAL *h_B;
  //REAL *h_C;
  //h_A = (REAL *)malloc(n2A * sizeof(h_A[0]));
  //if (h_A == 0)
  //{
  //    fprintf(stderr, "!!!! host memory allocation error (A)\n");
  //    Error();
  //}
  //h_B = (REAL *)malloc(n2B * sizeof(h_B[0]));
  //if (h_B == 0)
  //{
  //    fprintf(stderr, "!!!! host memory allocation error (B)\n");
  //    Error();
  //}
  //h_C = (REAL *)malloc(n2C * sizeof(h_C[0]));
  //if (h_C == 0)
  //{
  //    fprintf(stderr, "!!!! host memory allocation error (C)\n");
  //    Error();
  //}
  /* Fill the matrices with test data */
    // gemm ( transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c,
ldc )
    //   * C = alpha*A * B + beta * b
    //              mxn mxk        kxn
    //              lda     ldb  ldc
//  cublasHandle_t handle;
//  cublasStatus_t stat;
//  stat = cublasCreate(&handle);
//  if (stat != CUBLAS_STATUS_SUCCESS)
//  {
//      fprintf(stderr,"\n CuBLAS Initialization Failed \n");
//      cudaDeviceReset();
//      Error();
//  };
//  /* Allocate device memory for the matrices */
//  if (cudaMalloc((void **)&d_A, n2A * sizeof(d_A[0])) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! device memory allocation error (allocate
A)\n");
//      Error();
//  }
//  if (cudaMalloc((void **)&d_B, n2B * sizeof(d_B[0])) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! device memory allocation error (allocate
B)\n");
//      Error();
//  }
//  if (cudaMalloc((void **)&d_C, n2C * sizeof(d_C[0])) != cudaSuccess)
//  {
```

```
//      fprintf(stderr, "!!!! device memory allocation error (allocate
C)\n");
//      Error();
//  }
//  /* Initialize the device matrices with the host matrices */
//  stat = cublasSetVector(n2A, sizeof(A[0]), A, 1, d_A, 1);
//  if (stat != CUBLAS_STATUS_SUCCESS)
//  {
//      fprintf(stderr, "!!!! device access error (write A)\n");
//      Error();
//  }
//  stat = cublasSetVector(n2B, sizeof(B[0]), B, 1, d_B, 1);
//  if (stat != CUBLAS_STATUS_SUCCESS)
//  {
//      fprintf(stderr, "!!!! device access error (write B)\n");
//      Error();
//  }
//  stat = cublasSetVector(n2C, sizeof(C[0]), C, 1, d_C, 1);
//  if (stat != CUBLAS_STATUS_SUCCESS)
//  {
//      fprintf(stderr, "!!!! device access error (write C)\n");
//      Error();
//  }
//  printf("blas.h GEMM m=%i, n=%i, k=%i \n", dimy, dimx, dimk);
  //TRACE("-mkl- call gemm\n");
  //GEMM (CblasColMajor,CblasNoTrans, CblasNoTrans, &dimy, &dimx,
&dimk, &alpha, A, &dimy, B, &dimk, &beta, C, &dimy);
//  GEMM (CblasColMajor,CblasNoTrans, CblasNoTrans, dimy, dimx, dimk,
alpha, A, dimy, B, dimk, beta, C, dimy);

//  /* Performs operation using cublas */
//  stat = cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, dimy, dimx,
dimk, &alpha, d_A, dimy, d_B, dimk, &beta, d_C, dimy);
//  if (stat != CUBLAS_STATUS_SUCCESS)
//  {
//     fprintf(stderr, "!!!! kernel execution error.\n");
//     Error();
//  }
//  GEMM (CblasColMajor,CblasNoTrans, CblasNoTrans, dimy, dimx, dimk,
alpha, A, dimy, B, dimk, beta, C, dimy);
//  stat = cublasGetVector(n2A, sizeof(h_A[0]), d_A, 1, h_A, 1);
//   if (stat != CUBLAS_STATUS_SUCCESS)
//   {
//     fprintf(stderr, "!!!! device access error (read A)\n");
//     Error();
//   }
//   printf("%f h_A[0] data \n", h_A[0]);
//   printf("%f h_A[1] data \n", h_A[1]);
//   stat = cublasGetVector(n2B, sizeof(h_B[0]), d_B, 1, h_B, 1);
//    if (stat != CUBLAS_STATUS_SUCCESS)
//    {
//      fprintf(stderr, "!!!! device access error (read B)\n");
//      Error();
//    }
  /* Read the result back */
//  stat = cublasGetVector(n2C, sizeof(C[0]), d_C, 1, C, 1);
//  if (stat != CUBLAS_STATUS_SUCCESS)
```

```
//  {
//    fprintf(stderr, "!!!! device access error (read h_C)\n");
//    Error();
//  }
//  if (cudaFree(d_A) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! memory free error (d_A)\n");
//      Error();
//  }
//  if (cudaFree(d_B) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! memory free error (d_B)\n");
//      Error();
//  }
//  if (cudaFree(d_C) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! memory free error (d_C)\n");
//      Error();
//  }
//  /* Shutdown */
//  cublasDestroy(handle);
//  C = h_C;
  /* Memory clean up */
//  free(h_A);
//  free(h_B);
//  free(h_C);
//  fprintf(stderr, "Cuda blas.h GEMM Cycle Complete! \n");
}

#endif
```

D.8 cslm_train.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
```

```
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: cslm_train.cpp,v 1.4 2010/01/25 12:53:31 schwenk Exp $
 *
 * This is a simple program to perform the training of continuous space
LMs
 */

using namespace std;
#include <iostream>

#include "Mach.h"
#include "MachTab.h"
#include "MachTanh.h"
#include "MachSoftmax.h"
#include "MachSeq.h"
#include "MachPar.h"
#include "TrainerNgram.h"
#include "ErrFctSoftmCrossEntNgram.h"

int main (int argc, char *argv[])
{

   // get params
  if (argc != 13) {
    fprintf(stderr,"usage: %s train.df dev.df machine-name dim-proj
dim-hidden dim-output order bsize lrate-begin lrate-mult wdecay nb-
iter\n", argv[0]);
    Error();
  }

  int pdim=atoi(argv[4]);
  int hdim=atoi(argv[5]);
  int odim=atoi(argv[6]);
  int order=atoi(argv[7]);
  int bs=atoi(argv[8]);
  float lrb=atof(argv[9]);
  float lre=atof(argv[10]);
  float wd=atof(argv[11]);
  int nbit=atoi(argv[12]);

    // create projection layer
  MachPar mp;
  MachTab *mt = new MachTab(odim,pdim,bs);
  mt->TableRandom(0.1);
  mp.MachAdd(mt);
  REAL *tab_adr=mt->GetTabAdr();
  for (int i=1; i<order-1; i++) {
    MachTab *mt = new MachTab(tab_adr,odim,pdim,bs);
    mp.MachAdd(mt);
  }

    // add estimation layer
  MachTanh *mh = new MachTanh((order-1)*pdim,hdim,bs);
  mh->WeightsRandom(0.1); mh->BiasRandom(0.1);
  MachSoftmax *mo = new MachSoftmax(hdim,odim,bs);
  mo->WeightsRandom(0.1); mo->BiasRandom(0.1);
```

```
  MachSeq mlp;
  mlp.MachAdd(&mp);
  mlp.MachAdd(mh); mlp.MachAdd(mo);
  mlp.Info();

  ErrFctSoftmCrossEntNgram errfct(mlp);
  TrainerNgram trainer(&mlp, &errfct, argv[1], argv[2], lrb, lre, wd,
nbit);
  //cout << "Initial perplexity: " << trainer.TestDev() << endl;
  trainer.TrainAndTest();

  ofstream fs;
  fs.open(argv[3],ios::binary);
  CHECK_FILE(fs,argv[3]);
  mlp.Write(fs);
  fs.close();

  mp.Delete();
  delete mh;
  delete mo;

  return 0;
}
```

D.9 Data.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Data.cpp,v 1.12 2010/01/25 12:53:31 schwenk Exp $
 */

using namespace std;
#include <iostream>
```

```
#include <stdlib.h>

#include "Tools.h"
#include "Data.h"
#include "DataAscii.h"
#include "DataNgramBin.h"

const int   DATA_LINE_LEN=1024;
const char* DATA_HEADER_TXT="DataDescr";
const int   DATA_HEADER_ID=1;
const char* DATA_PRELOAD="Preload";
const int   DATA_PRELOAD_ALWAYS=1;  // order of constants is important
!
const int   DATA_PRELOAD_ONCE=2;
const int   DATA_PRELOAD_DONE=3;
const char* DATA_RESAMPL_MODE="ResamplMode";
const char* DATA_RESAMPL_SEED="ResamplSeed";
const char* DATA_SHUFFLE_MODE="ShuffleMode";
const char* DATA_NORMALIZE_MODE="Normalize";

/*************************
 *
 *************************/

Data::Data(char *p_fname)
 : fname(p_fname), idim(0), odim(0), nb_totl(0),
   preload(0), resampl_mode(0), resampl_seed(1234567890),
shuffle_mode(0),
   norm_mode(0),
   idx(-1), mem_inp(NULL), mem_trg(NULL), input(NULL), target(NULL)
{

  cout << "Opening data description '" << fname << "'" << endl;
  ifstream ifs;
  ifs.open(fname,ios::in);
  CHECK_FILE(ifs,fname);

    // parsing data description
  int i=ReadInt(ifs,DATA_HEADER_TXT);
  if (i!=DATA_HEADER_ID) Error("unknown data description header\n");

  while (!ifs.eof()) {
    bool ok=false;
    string buf; char line[DATA_LINE_LEN];
    ifs >> buf;
    if (buf[0]=='#') {ifs.getline(line, DATA_LINE_LEN); continue;} //
skip comments
    if (buf=="") break; // HACK
    if (buf==DATA_PRELOAD) { preload=DATA_PRELOAD_ALWAYS; ok=true; }
    if (buf==DATA_RESAMPL_MODE) { ifs >> resampl_mode; ok=true; }
    if (buf==DATA_RESAMPL_SEED) { ifs >> resampl_seed; ok=true; }
    if (buf==DATA_SHUFFLE_MODE) { ifs >> shuffle_mode; ok=true; }
    if (buf==DATA_NORMALIZE_MODE) { ifs >> norm_mode; ok=true; }
    if (buf==DATA_FILE_ASCII) {
      datafile.push_back(new DataAscii(ifs)); ok=true;
    }
    if (buf==DATA_FILE_NGRAMBIN) {
```

```
      datafile.push_back(new DataNgramBin(ifs)); ok=true;
    }
    if (datafile.size()==1) {idim=datafile[0]->GetIdim();
odim=datafile[0]->GetOdim(); }
    if (datafile.size()>=1) {
      if (idim != datafile.back()->GetIdim()) Error("mismatch in input
dimension\n");
      if (odim != datafile.back()->GetOdim()) Error("mismatch in output
dimension\n");
    }

    if (!ok) {
      ifs.getline(line, DATA_LINE_LEN);
      cerr << buf << "" << line << endl;
      Error("parse error in above line of the datafile\n");
    }
  }
  ifs.close();

  nb_totl=0;
  cout << "Summary of used data:" << endl;
  for (i=0; i<(int) datafile.size(); i++) nb_totl+=datafile[i]->Info();

  cout << " - total number of examples: " << nb_totl << endl;
  if (resampl_mode) {
    cout << " - resampling with seed " << resampl_seed << endl;
    srand48(resampl_seed);
  }
  if (preload > 0) {
    mem_inp = new REAL[nb_totl*idim];
    if (odim>0) mem_trg = new REAL[nb_totl*odim];

      // check whether there is a resmpling coeff != 0
      // i.e. we need to resample at each rewind
    float s=0;
    for (vector<DataFile*>::iterator it = datafile.begin();
it!=datafile.end(); ++it)
      s+=(*it)->GetResampl();
    if (s>=datafile.size()) {
      preload=DATA_PRELOAD_ONCE;
      cout << " - all resampling coefficients are set to one, loading
data once\n";
    }

  }
  else {
    if (norm_mode>0)
      Error("Normalization of the data is only implemented with
preloading\n");
  }
  Preload();
  Shuffle();
}

/*************************
 *
 *************************/
```

```
Data::Data(DataFile &df)
 : fname(NULL), idim(df.GetIdim()), odim(df.GetOdim()),
nb_totl(df.GetNbex()),
   preload(0), resampl_mode(0), resampl_seed(1234567890),
shuffle_mode(0),
   norm_mode(0),
   idx(-1), mem_inp(NULL), mem_trg(NULL), input(NULL), target(NULL)
{

  datafile.push_back(&df);
}

Data::~Data()
{
  if (preload) {
    delete [] mem_inp;
    if (odim>0) delete [] mem_trg;
  }
  for (vector<DataFile*>::iterator it = datafile.begin();
it!=datafile.end(); ++it)
      delete (*it);
  datafile.clear();
}

/*************************
 *
 *************************/

void Data::Shuffle()
{
  if (shuffle_mode < 1 || !preload) return;

  REAL      *inp = new REAL[idim];
  REAL      *trg = new REAL[odim];

  cout << " - shuffling data " << shuffle_mode << " times ...";
  cout.flush();
  for (int i=0; i<shuffle_mode*nb_totl; i++) {
    int i1 = (int) (nb_totl * drand48());
    int i2 = (int) (nb_totl * drand48());

    memcpy(inp, mem_inp + i1*idim, idim*sizeof(REAL));
    memcpy(mem_inp + i1*idim, mem_inp + i2*idim, idim*sizeof(REAL));
    memcpy(mem_inp + i2*idim, inp, idim*sizeof(REAL));

    if (odim>0) {
      memcpy(trg, mem_trg + i1*odim, odim*sizeof(REAL));
      memcpy(mem_trg + i1*odim, mem_trg + i2*odim, odim*sizeof(REAL));
      memcpy(mem_trg + i2*odim, trg, odim*sizeof(REAL));
    }

  }

  delete [] inp; delete [] trg;
  cout << " done" << endl;
}
```

```
//*************************
//
//

void Data::Preload()
{
  if (!preload || preload>DATA_PRELOAD_ONCE) return;
  if (preload == DATA_PRELOAD_ONCE) preload=DATA_PRELOAD_DONE;

  cout << " - loading all data into memory" << endl;

  int idx=0;
  for (vector<DataFile*>::iterator it = datafile.begin();
it!=datafile.end(); ++it) {
    (*it)->Rewind();
    int n = -1, maxn = (*it)->GetNbresampl();
//cout << "Resampl " << maxn << " examples from file into " <<  (*it)-
>input << endl;
    while (++n < maxn) {
      (*it)->Resampl();
//cout << "n: " << n << ", idx: " << (*it)->idx << endl;
      memcpy(mem_inp+idx*idim, (*it)->input, idim*sizeof(REAL));
      if (odim > 0) memcpy(mem_trg+idx*odim, (*it)->target_vect,
odim*sizeof(REAL));
      idx++;
    }
  }

  if (norm_mode & 1) {
    cout << " - normalizing input: subtract mean" << endl;
    for (int i=0; i<idim; i++) {
      int e;
      REAL m=0, *mptr;
      for (e=0, mptr=mem_inp+i; e<idx; e++, mptr+=idim) m+=*mptr;
      m = m/idx; // mean
      for (e=0, mptr=mem_inp+i; e<idx; e++, mptr+=idim) *mptr -= m;
    }
  }

  if (norm_mode & 2) {
    cout << " - normalizing input: divide by variance" << endl;
    for (int i=0; i<idim; i++) {
      int e;
      REAL m=0, m2=0, *mptr;
      for (e=0, mptr=mem_inp+i; e<idx; e++, mptr+=idim) { m+=*mptr;
m2+=*mptr * *mptr; }
      m = m/idx;  // mean
      m2 = m2/idx - m; // var = 1/n sum_i x_i^2  -  mu^2
      if (m2>0)
        for (e=0, mptr=mem_inp+i; e<idx; e++, mptr+=idim)
          *mptr = (*mptr - m) / m2;
    }
  }
#ifdef DEBUG
    for (int e=0; e<idx; e++) {
```

```
      for (int i=0; i<idim; i++) printf(" %5.2f",mem_inp[e*idim+i]);
printf("\n");
    }
#endif
}


/*************************
 *
 *************************/

void Data::Rewind()
{
  if (preload) {
       // clear all data, resample and shuffle again
    Preload();
    Shuffle();
  }
  else {
    for (vector<DataFile*>::iterator it = datafile.begin();
it!=datafile.end(); ++it) (*it)->Rewind();
  }
  idx = -1;
}

/*************************
 * Advance to next data
 *************************/

bool Data::Next()
{
  if (idx >= nb_totl-1) return false;
  idx++;

  if (preload) {
      // just advance to next data in memory
    input = &mem_inp[idx*idim];
    if (odim>0) target = &mem_trg[idx*odim];
//printf("DATA:"); for (int i =0; i<idim; i++) printf(" %5.2f",
input[i]); printf("\n");
    return true;
  }

  if (shuffle_mode > 0) {
      // resample in RANDOMLY SELECTED datafile until data was found
      // we are sure to find something since idx was checked before
    int df = (int) (drand48() * datafile.size());
//cout << " df=" << df << endl;
    datafile[df]->Resampl();
    input = datafile[df]->input;
    if (odim>0) target = datafile[df]->target_vect;
  }
  else {
      // resample SEQUENTIALLY all the data files
    static int df=0, i=-1, nbdf=datafile[df]->GetNbex();
    if (idx==0) {df = 0, i=-1, nbdf=datafile[df]->GetNbex(); }    //
(luint) this) is a hack to know when there was a global rewind
```

```
    if (++i >= nbdf) { df++; nbdf=datafile[df]->GetNbex(); i=-1; }
    if (df >= (int) datafile.size()) Error("internal error: no examples
left\n");
//printf("seq file: df=%d, i=%d\n", df,i);
    datafile[df]->Resampl();  //TODO: idx= ??
//cout << " got df=" << df << " idx="<<idx<<endl;
    input = datafile[df]->input;
    if (odim>0) target = datafile[df]->target_vect;
  }

  return true;
}
```

D.10 Data.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Data.h,v 1.8 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Data_h
#define _Data_h

#include <iostream>
#include <fstream>
#include <vector>
#include "Tools.h"
#include "DataFile.h"

// Names of information in files

extern const int   DATA_LINE_LEN;
extern const char* DATA_HEADER_TXT;
```

```
extern const int   DATA_HEADER_ID;
extern const char* DATA_PRELOAD;
extern const char* DATA_RESAMPL_MODE;
extern const char* DATA_RESAMPL_SEED;
extern const char* DATA_SHUFFLE_MODE;

/*
 * Strategie
 *  - there is one funtion Rewind() and Next() which should not be
overriden
 *  - they perform all the processing with preloading, shuffling, etc
 *  - the class specific processing is done in First() and Advance()
 */


class Data
{
protected:
  char *fname;
  int  idim, odim;      // dimensions
  int  nb_totl;         // number of examples
    // flags
  int preload;          //
  int resampl_mode;     //
  int resampl_seed;     //
  int shuffle_mode;     //
  int norm_mode;  // evtl. perform normalization; bits: 1=substract
mean, 2=devide by var.
    // data files
  vector<DataFile*>     datafile;
    // actual data
  int  idx;       // index of current example [0,nb-1]
  REAL *mem_inp;  // all the input data in memory
  REAL *mem_trg;  // all the output data in memory
    // local tools, only used when prelaod is activated
  void Preload(); // preload all data
  void Shuffle(); // shuffle in memory
public:
  Data(char *fname);
  Data(DataFile&);      // simplified version with one Datafile only
  ~Data();
    // access function to local variables
  char *GetFname() {return fname;}
  int GetIdim() {return idim;}
  int GetOdim() {return odim;}
  int GetNb() {return nb_totl;}
  int GetIdx() {if (idx<0) Error("DataNext() must be called before
GetIdx()"); return idx;};
    // the following two pointers are only valid after first DataNext()
!
  REAL *input;          // pointer to current inputs
  REAL *target;         // pointer to current target
  //REAL *GetData() {return val;}
    // main functions to access data
  void Rewind();  // rewind to first example, performs resmplaing,
shuffling etc if activated
```

```
  bool Next();          // advance to next example, return FALSE if at
end
};

#endif
```

D.11 DataAscii.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataAscii.cpp,v 1.8 2010/01/25 12:53:31 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "Data.h"
#include "DataAscii.h"

const char* DATA_FILE_ASCII="DataAscii";

DataAscii::DataAscii(ifstream &ifs) : DataFile::DataFile(ifs)
{

  dfs.open(fname,ios::in);
  CHECK_FILE(dfs,fname);
  char buf[DATA_LINE_LEN];
  dfs.getline(buf,DATA_LINE_LEN);
  sscanf(buf, "%d %d %d", &nbex, &idim, &odim);
  printf(" - %s: ASCII data with %d examples of dimension %d -> %d\n",
fname, nbex, idim, odim);
```

```
  if (idim>0) input = new REAL[idim];
  if (odim>0) target_vect = new REAL[odim];
}


/*************************
 *
 *************************/

DataAscii::~DataAscii()
{
  dfs.close();
  if (idim>0) delete [] input;
  if (odim>0) delete [] target_vect;
}


/*************************
 *
 *************************/

void DataAscii::Rewind()
{
  // dfs.seekg(0,ios::beg); HACK: does not work
  dfs.close();
  dfs.open(fname,ios::in);
  CHECK_FILE(dfs,fname);
  char buf[DATA_LINE_LEN];
  dfs.getline(buf,DATA_LINE_LEN);
}

/*************************
 *
 *************************/

bool DataAscii::Next()
{
  char line[DATA_LINE_LEN];
  dfs.getline(line, DATA_LINE_LEN);
  if (dfs.eof()) return false;
          else idx++;

    // parse input data
  char *lptr=line;
//cout << "\nLINE: " << line << endl;
  for (int i=0; i<idim; i++) {
//cout << "parse:" <<lptr<<"; ";
    while (*lptr==' ' || *lptr=='\t') lptr++;
    if (!*lptr) Error("incomplete input in ASCII datafile");
    if (sscanf(lptr, "%f", input+i)!=1) Error("parsing source in ASCII
datafile");
//cout << "got i[" <<i << "] " << input[i] << endl;
    while (*lptr!=' ' && *lptr!='\t' && *lptr!=0) lptr++;
  }

  if (odim<=0) return true;
```

```
    // parse target data
  for (int i=0; i<odim; i++) {
//cout << "parse:" <<lptr<<"; ";
    while (*lptr==' ' || *lptr=='\t') lptr++;
    if (!*lptr) Error("incomplete target in ASCII datafile");
    if (sscanf(lptr, "%f", target_vect+i)!=1) Error("parsing target in
ASCII datafile");
//cout << "got t[" <<i << "] " << target_vect[i] << endl;
    while (*lptr!=' ' && *lptr!='\t' && *lptr!=0) lptr++;
  }

  return true;
}
```

D.12 DataAscii.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataAscii.h,v 1.4 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _DataAscii_h
#define _DataAscii_h

#include <iostream>
#include <fstream>

#include "DataFile.h"

extern const char* DATA_FILE_ASCII;

class DataAscii : public DataFile
```

```
{
protected:
  ifstream dfs;
public:
  DataAscii(ifstream &ifs);
  virtual ~DataAscii();
  virtual void Rewind();
  virtual bool Next();
};

#endif
```

D.13 DataFile.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataFile.cpp,v 1.6 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
#include "Data.h"
#include "DataFile.h"

DataFile::DataFile(ifstream &ifs)
 : idim(0), odim(0), nbex(0), resampl_coeff(1.0), fname(NULL),
   idx(-1), input(NULL), target_vect(NULL)
{
  char p_fname[DATA_LINE_LEN];
```

```
  ifs >> p_fname >> resampl_coeff;
  if (resampl_coeff<=0 || resampl_coeff>1)
    Error("resampl coefficient must be in (0,1]\n");
  fname=strdup(p_fname);

  // memory allocation of input and target_vect should be done in
subclass
  // in function of the dimension and number of examples
}

DataFile::DataFile(char *p_fname, float p_rcoeff)
 : idim(0), odim(0), nbex(0), resampl_coeff(p_rcoeff),
fname(strdup(p_fname)),
   idx(-1), input(NULL), target_vect(NULL)
{

  // memory allocation of input and target_vect should be done in
subclass
  // in function of the dimension and number of examples
}

DataFile::~DataFile()
{
  if (fname) free(fname);
  // memory deallocation of input and target_vect should be done in
subclass

}

/*************************
 *
 *************************/

int DataFile::Info()
{
  int nbr=resampl_coeff*nbex;
  printf(" - %s  %6.4f * %9d = %9d\n", fname, resampl_coeff, nbex,
nbr);
  return nbr;
}

/*************************
 *
 *************************/

void DataFile::Rewind()
{
  Error("DataFile::Rewind() should be overriden");
}

//*************************
// read next data in File
// Return false if EOF

bool DataFile::Next()
{
```

```
  Error("DataFile::Next() should be overriden");
  return false;
}

//*************************
// generic resampling function using sequential file reads
// cycles sequentially through data until soemthing was found
// based on DataNext() which may be overriden by subclasses
// returns idx of current example

int DataFile::Resampl()
{
  bool ok=false;

  while (!ok) {
    if (!Next()) Rewind(); // TODO: deadlock if file empty
//cout << "Resampled: ";
//for (int i=0; i<idim; i++) cout << input[i] << " ";
    ok = (drand48() < resampl_coeff);
//cout << " ok=" << ok << endl;
  }

  return idx;
}
```

D.14 DataFile.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataFile.h,v 1.5 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _DataFile_h
```

```
#define _DataFile_h

#include <iostream>
#include <fstream>
#include <vector>
#include "Tools.h"

class DataFile {
protected:
  int idim, odim, nbex;
  float resampl_coeff;
    // internal handling of data
  char *fname;
public:
    // current data
  int  idx;
  REAL *input;          // current input data
  REAL *target_vect;    //         output data
  int  target_id; //         index of output [0..odim)
   // functions
  DataFile(ifstream &ifs);
  DataFile(char *, float =1.0);
  virtual ~DataFile();
   // access function
  int GetIdim() { return idim; }
  int GetOdim() { return odim; }
  int GetNbex() { return nbex; }
  int GetNbresampl() { return (int) (nbex*resampl_coeff); }
  float GetResampl() { return resampl_coeff; }
   // main interface
  virtual int Info();         // display line with info after loading
the data
  virtual void Rewind();      // rewind to first element
  virtual bool Next();        // advance to next data
  virtual int Resampl();      // resample another data (this may skip
some elements in the file)
};

#endif
```

D.15 DataNgramBin.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
```

```
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataNgramBin.cpp,v 1.11 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>

// system headers
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include "Tools.h"
#include "Data.h"
#include "DataNgramBin.h"

const char* DATA_FILE_NGRAMBIN="DataNgramBin";
const int DATA_NGRAM_IGN_BOS=1;
const int DATA_NGRAM_IGN_UNK=2;
const int DATA_NGRAM_IGN_UNKall=4;
const int DATA_NGRAM_IGN_EOS=8;     // TODO: not implemented
const int DATA_NGRAM_IGN_ALL=15;


//*******************

void DataNgramBin::do_constructor_work()
{
    // parse header binary Ngram file
  fd=open(fname, O_RDONLY);
  if (fd<0) {
    perror(fname); Error();
  }
  read(fd, &nbl, sizeof(int));
  read(fd, &nbex, sizeof(int));
  read(fd, &vocsize, sizeof(int));
  int s;
  read(fd, &s, sizeof(int));
  if (s != sizeof(WordID)) {
    fprintf(stderr,"binary n-gram data uses %d bytes per index, but
this code is compiled for %d byte indices\n", s, (int) sizeof(WordID));
    Error();
  }
  read(fd, &bos, sizeof(WordID));
  read(fd, &eos, sizeof(WordID));
```

```
  read(fd, &unk, sizeof(WordID));
  printf(" - %s binary ngram file with %d words in %d lines, order=%d,
mode=%d\n", fname, nbex, nbl, order, mode);

  idim=order-1;
  odim=1;

  if (idim>0) {
    input = new REAL[idim];
    wid = new WordID[order];
    for (int i=0; i<order; i++) wid[i]=bos;
  }
  target_vect = new REAL[odim];

    // counting nbex to get true number of examples
  cout << "    counting ..."; cout.flush();
  int n=0;
  nbs=nbw=nbu=nbi=0;
  while (DataNgramBin::Next()) n++;
  printf(" %d %d-grams (%d unk, %d ignored)\n", n, order, nbu, nbi);
  if (n>nbex)
    Error("Number of counted examples is larger than information in
file header !?");
  nbex=n;  //
}

//*******************

DataNgramBin::DataNgramBin(ifstream &ifs) : DataFile::DataFile(ifs),
  order(4), mode(0), nbw(0), nbs(0), nbu(0), nbi(0)
{
    // DataNgramBin <file_name> <resampl_coeff> <order> [flags]
    // parse addtl params
  ifs >> order >> mode;
  if (order<2 || order>9)
    Error("order must be in [2,9]\n");
  if (mode<0 || mode>DATA_NGRAM_IGN_ALL)
    Error("wrong value of DataNgramBin mode\n");

  do_constructor_work();
}

//*******************

DataNgramBin::DataNgramBin(char *p_fname, float p_rcoeff, int p_order,
int p_mode)
  : DataFile::DataFile(p_fname, p_rcoeff),
  order(4), mode(p_mode), nbw(0), nbs(0), nbu(0), nbi(0)
{

  do_constructor_work();
    // skip counting for efficieny reasons
  nbw=nbex;       // this should be an upper bound on the number of n-
grams
}

//*******************
```

```
DataNgramBin::~DataNgramBin()
{

  close(fd);
  if (idim>0) {
    delete [] wid;
    delete [] input;
  }
  delete [] target_vect;
}


//*******************

bool DataNgramBin::Next()
{
  bool ok=false;
  int i;

   // we may need to skip some n-grams in function of the flags
  while (!ok) {

      // read from file into, return if EOF
    WordID w;
    if (read(fd, &w, sizeof(w)) != sizeof(w)) return false;
//printf("read: %d\n",w);

      // shift previous order
    for (i=1; i<order; i++) wid[i-1]=wid[i];
    wid[order-1]=w;
//printf(" wid: %d %d %d\n",wid[0],wid[1],wid[2]);

     // update statistics
    if (w == bos) ; /* nothing to count */
    else if (w == eos) nbs++;
    else if (w == unk) nbu++;
    else nbw++;

     // check if n-gram is valid according to the selected mode

    if (w == bos) {
        // new BOS, initialize the whole order to BOS
        // (it will be shifted away)
      for (i=0; i<order; i++) wid[i] =  bos;
//printf("skip [new bos]\n");
      continue;
    }

    if (mode & DATA_NGRAM_IGN_UNK) {
        // ignore n-grams with <UNK> at last position
      if (w == unk) {
        nbi++;
//printf("skip [predict unk]\n");
        continue;
      }
    }
```

```
    if (mode & DATA_NGRAM_IGN_UNKall) {
        // ignore n-grams that contain <UNK> anywhere
      for (i=0; i<order-1; i++) {
        if (wid[i] == unk) {
          nbi++;
//printf("skip [any unk]\n");
          break;
        }
      }
      if (i < order-1) continue;
    }

    if (mode & DATA_NGRAM_IGN_BOS) {
        // ignore n-grams that contain <BOS> elsewhere than at 1st
position
      for (i=1; i<order; i++)
        if (wid[i] == bos) {
          nbi++;
//printf("skip [bos]\n");
          break;
      }
      if (i < order) continue;
    }

      /* standard mode */
    ok=true;
  } // of while (!ok)

//printf("keep: %d %d %d\n",wid[0],wid[1],wid[2]);
  for (i=0; i<order-1; i++) input[i] = (REAL) wid[i];       // careful:
we cast to float which may give
  target_vect[0] = (int) wid[i];                       // rounding
problems of the integers
  target_id = (int) wid[i];

  idx++;
  return true;
}

/********************
 *
 ********************/


int DataNgramBin::Info()
{
  return DataFile::Info();
  //int nbr=resampl_coeff*nbex;
  //printf(" - %s  %6.4f * %9d = %9d [ngram order=%d, mode=%d, unk=%d,
bos=%d, eos=%d]\n", fname, resampl_coeff, nbex, nbr, order, mode, unk,
bos, eos);
  //return nbr;
}

void DataNgramBin::Rewind()
{
```

```
  lseek(fd,
sizeof(nbl)+sizeof(nbex)+sizeof(vocsize)+sizeof(int)+3*sizeof(WordID),
SEEK_SET);
  idx=-1;
}
```

D.16 DataNgramBin.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: DataNgramBin.h,v 1.4 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _DataNgramBin_h
#define _DataNgramBin_h

#include <iostream>
#include <fstream>

#include "DataFile.h"

extern const char* DATA_FILE_NGRAMBIN;

typedef int WordID;

// Syntax of a line in data description:
// DataNgramBin <file_name> <resampl_coeff> <order> [flags]
//  u: skip n-grams with <unk> at the right most position
//  U: skip n-grams with <unk> anywhere
//  b: skip n-grams with <s> elsewhere than at the left most position
//  e: skip n-grams with </s> elsewhere than at the right most position

class DataNgramBin : public DataFile
```

```
{
private:
  void do_constructor_work();
protected:
  int fd;         // UNIX style binary file
  int vocsize;          // vocab size (including <s>, </s> and <unk>)
  int order;            // order of the ngrams
  int mode;       // see above for possible flags
  WordID *wid;          // whole n-gram context
  WordID bos, eos, unk; // word ids of special symbols
    // stats (in addition to nbex in mother class)
  int  nbl, nbw, nbs, nbu;// lines, words, sentences, unks
  int  nbi;       // ignored n-grams
public:
  DataNgramBin(ifstream &ifs);
  DataNgramBin(char*, float =1.0, int =4, int =3);
  virtual ~DataNgramBin();
  virtual int Info();
  virtual bool Next();
  virtual void Rewind();
};

#endif
```

D.17 ErrFct.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFct.cpp,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
```

```
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "ErrFct.h"
#include "cuda_runtime.h"

ErrFct::ErrFct (Mach &mach)
 : dim(mach.GetOdim()), bsize(mach.GetBsize()),
   output(mach.GetDataOut()), target(NULL)//, grad(new REAL[dim*bsize])
//   output(mach.GetDataOut()), target(NULL), grad(new REAL[dim*bsize])
{
//cerr << "Constructor ErrFct: alloc gradient of size " << dim << endl;
      cudaMallocManaged(&grad, dim*bsize*sizeof(REAL));

}

//*********************************************************************
****************

REAL ErrFct::CalcValue(int eff_bsize) { return 0; }

REAL ErrFct::CalcGrad(int eff_bsize) {
  if (eff_bsize<=0) eff_bsize=bsize;
  for (int i=0; i<dim*eff_bsize; i++) grad[i]=0.0;
  return 0;
}

ErrFct::~ErrFct()
{
  cudaFree(grad);
}
```

D.18 ErrFct.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFct.h,v 1.6 2010/01/25 12:27:07 schwenk Exp $
 *
 * Class definiton of a general error funtion
 */

#ifndef _ErrFct_h
#define _ErrFct_h

#include <iostream>
#include "Tools.h"
#include "Mach.h"
#include "Data.h"


class ErrFct
{
private:
protected:
  int dim;              // output dimension of machine
  int bsize;
  REAL  *output;        // pointer to output data (stored in machine)
  REAL  *target;        // pointer to target data (stored in trainer)
  REAL  *grad;                // calculated gradient (stored in this
class)
public:
  ErrFct(Mach&);
  virtual ~ErrFct();// { delete [] grad; }
  void SetOutput(REAL *p_output) {output=p_output; }
  void SetTarget(REAL *p_target) {target=p_target; }
  REAL *GetGrad() {return grad; };
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                 // calculate NEGATIF
gradient of error function
};

#endif
```

D.19 ErrFctCrossEnt.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
```

```
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctCrossEnt.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 *
 * Class definiton of cross entropy error function
 *   E = sum_i  d_i * ln o_i
 *   dE/do_k = d_k / o_k   for o_k <> 0
 * This is usually used with softmax outputs
 */

#ifndef _ErrFctCrossEnt_h
#define _ErrFctCrossEnt_h

#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctCrossEnt : public ErrFct
{
public:
  ErrFctCrossEnt(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                // calculate NEGATIF
gradient of error function
};

#endif
```

D.20 ErrFctCrossEntNgram.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
```

```
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctCrossEntNgram.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 * Class definiton of cross entropy error function
 * Spezial version for NNs that predict words
 *  - the NN has a large output dimension (vocsize or limited to
shorlist)
 *  - the data has one dimensional targets that are taken as index into
 *    the word list
 *  - therfore the target vector is binary: 1 at the position of the to
predicted
 *    word, 0 elsewhere
 *
 *   E = sum_i  d_i * ln o_i
 *   dE/do_k = d_k / o_k   for o_k <> 0
 * This is usually used with softmax outputs
 */

#ifndef _ErrFctCrossEnt_h
#define _ErrFctCrossEnt_h

using namespace std;
#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctCrossEnt : public ErrFct
{
private:
  int voc_size;         //
                  // the private var "dim" is set to 1
public:
  ErrFctCrossEnt(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                 // calculate NEGATIF
gradient of error function
};

#endif
```

D.21 ErrFctMCE.h

```
/*
```

```
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctMCE.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 *
 * Class definiton of ``mean classification error'' function (MCE)
 *   we use MSE for training, but the value of the error function is
 *   the percentage of wrongly classified examples
 */

#ifndef _ErrFctMCE_h
#define _ErrFctMCE_h

#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctMCE : public ErrFct
{
public:
  ErrFctMCE(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                // calculate NEGATIF
gradient of error function
};

#endif
```

D.22 ErrFctMSE.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
```

```
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctMSE.h,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 *
 * Class definiton of mean squared error error function (MSE)
 *   E = sum_i  (o_i - d_i)^2
 *   dE/do_k = 2 (o_k - d_k)
 */

#ifndef _ErrFctMSE_h
#define _ErrFctMSE_h

#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctMSE : public ErrFct
{
public:
  ErrFctMSE(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);          // Calculate value of error
function
  virtual REAL CalcGrad(int=0);                // calculate NEGATIF
gradient of error function
};

#endif
```

D.23 ErrFctSoftmCrossEntNgram.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
```

```
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctSoftmCrossEntNgram.cpp,v 1.5 2010/01/25 12:27:07 schwenk
Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "ErrFctSoftmCrossEntNgram.h"
#include "Gpu.cuh"
#include "cuda_runtime.h"

//*********************************************************************
*************
// E = log(sum_i d_i ln o_i)
//   = ln o_t     where t is the target index
//   output: dimension voc_size
//   target: dimension 1 with values [0,voc_size[
// We also take the log since this can't be done later if bsize>1

REAL ErrFctSoftmCrossEntNgram::CalcValue(int eff_bsize) {
  REAL *optr=output;
  REAL *tptr=target;
  double err=0.0;
//  cudaDeviceSynchronize();
  if (eff_bsize<=0) eff_bsize=bsize;
//  for (int b=0; b<eff_bsize; b++) {
//    if (*tptr<0 || *tptr>=dim) {
//      printf("ErrFctSoftmCrossEntNgram::CalcValue(): target out of
bounds (%d) must be in [0,%d[\n",(uint)*tptr,dim);
//      Error();
//    }
////printf("b=%d, tidx=%f, out=%f\n", b, *tptr, optr[(uint) *tptr]);
//    err += log(optr[(uint) *tptr++]);
////printf("err=%f\n",err);
//    optr += dim;
```

```
//  }
  err = GpuErrFctSoftmCrossEntNgramCalcValue(eff_bsize, dim, output,
target);
  return (REAL) err; // TODO: normalize ?
}

// We include here the derivation of the softmax outputs since we have
//   dE/da_k = sum_i dE/do_i do_i/da_k
// Due to the sum, dE/do_i and do_i/da_k can't be calculated separately
// dE/do_i = d_i/o_i
// do_i/da_k = o_i (kronecker_ik - o_k)
//  -> dE/da_k = sum_i d_i/o_i * o_i (kronecker_ik - o_k)
//             = sum_i d_i (kronecker_ik - o_k)
//             = (kronecker_tk - o_k)       since d_i=0 for i!=t
REAL ErrFctSoftmCrossEntNgram::CalcGrad(int eff_bsize) {
//  REAL *optr=output;
//  REAL *tptr=target;
//  REAL *gptr=grad;
  REAL err=0.0;
  uint      tidx;
//  cudaDeviceSynchronize();
//  printf("Entered ErrFctSoftmCrossEntNgram::CalcGrad.... \n");
  if (eff_bsize<=0) eff_bsize=bsize;
  err=GpuErrFctSoftmCrossEntNgramCalcGrad(eff_bsize, dim, output, grad,
target);
//  for (int b=0; b<eff_bsize; b++) {
//    for (int i=0; i<dim; i++) gptr[i] = -optr[i];
//    tidx=(uint) *tptr++;
//    if (tidx<0 || tidx>=(uint) dim) {
//      printf("ErrFctSoftmCrossEntNgram::CalcGrad(): target out of
bounds (%d) must be in [0,%d[\n",tidx,dim);
//      Error();
//    }
//    err += log(optr[tidx]);
//    gptr[tidx] += 1.0;
//    gptr+=dim; optr+=dim;
//  }
//  cudaDeviceSynchronize();
//  printf("ErrFctSoftmCrossEntNgram::CalcGrad complete!!! \n");
  return (REAL) err; // TODO: normalize ?
}
```

D.24 ErrFctSoftmCrossEntNgram.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
```

```
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: ErrFctSoftmCrossEntNgram.h,v 1.4 2010/01/25 12:27:07 schwenk
Exp $
 *
 * Class definiton of cross entropy error function
 * Spezial version for NNs that predict words
 *  - the NN has a large output dimension (vocsize or limited to
shorlist)
 *  - the data has one dimensional targets that are taken as index into
 *    the word list
 *  - therefore the target vector is binary: 1 at the position of the
to predicted
 *    word, 0 elsewhere
 *
 *   E = sum_i  d_i * ln o_i
 *   dE/do_k = d_k / o_k   for o_k <> 0
 * This is usually used with softmax outputs
 */

#ifndef _ErrFctSoftmCrossEnt_h
#define _ErrFctSoftmCrossEnt_h

#include <iostream>
#include "Tools.h"
#include "ErrFct.h"


class ErrFctSoftmCrossEntNgram : public ErrFct
{
private:
  int voc_size;         //
                  // the private var "dim" is set to 1
public:
  ErrFctSoftmCrossEntNgram(Mach &mach) : ErrFct(mach) {};
  virtual REAL CalcValue(int=0);    // Calculate value of error
function
  virtual REAL CalcGrad(int=0);           // calculate NEGATIF gradient
of error function
};

#endif
```

D.25 Eval.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Eval.h,v 1.2 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Eval_h
#define _Eval_h

#include <iostream>
#include "Tools.h"
#include "Mach.h"
#include "Data.h"


class Eval
{
private:
protected:
  Mach  &mach;                // network to evaluate
  int idim, odim, bsize;      // copied here for faster access
    // buffer to store bsize examples
  REAL  *buf_input;
  REAL  *buf_target;
public:
  Eval(Mach&, int=16384);
  virtual ~Eval();
  virtual void Data(Data &data, int* = NULL);   // evaluate on existing
data
  virtual void BlockEval(WordId &wid, int order, float *p, int n);
  virtual void BlockFinish();
};
```

```
#endif
```

D.26 EvalNgramBin.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: EvalNgramBin.cpp,v 1.4 2010/01/26 19:37:22 schwenk Exp $
 */

using namespace std;

#include "Tools.h"
#include "EvalNgramBin.h"

#include <algorithm>
#include "cuda_runtime.h"

EvalNgramBin::EvalNgramBin(Mach &p_mach, uint p_max_req)
 : mach(p_mach), max_req(p_max_req)
{

  idim=mach.GetIdim(); odim=mach.GetOdim(); bsize=mach.GetBsize();

  if (odim < 16) {
    fprintf(stderr,"EvalNgramBin: output dimension of the machine is
suspiciously small (%d)\n", odim);
    Error();
  }

  //buf_input = new REAL[idim*bsize];
  cudaMallocManaged(&buf_input, idim*bsize*sizeof(REAL));
  mach.SetDataIn(buf_input);
}
```

```
EvalNgramBin::~EvalNgramBin()
{
  for (vector<NgramReq*>::iterator it=req.begin(); it<req.end(); ++it)
delete *it;
  //delete [] buf_input;
  cudaFree(buf_input);
}

REAL EvalNgramBin::Eval(WordID *wid, int order, float *p)
{
  if (order-1 != idim) {
    fprintf(stderr,"EvalNgramBin::Eval(): requested context size (%d)
does not match input dimension of neural network (%d)\n", order-1,
idim);
    Error();
  }
  int oidx= wid[order-1];
  if (oidx<0 || oidx>=odim) {
    fprintf(stderr,"EvalNgramBin::Eval(): wrong index of the predicted
word (%d), should be in [0,%d[\n", oidx, odim);
    Error();
  }

  for (int i=0; i<order-1; i++) buf_input[i]=(REAL) wid[i];
#ifdef DEBUG
  for (int i=0; i<order-1; i++) printf(" %d", wid[i]);
  printf(" -> %d\n", oidx);
#endif
  mach.Forw(1);

  if (p) *p=mach.GetDataOut()[oidx];

  return mach.GetDataOut()[oidx];
}

void EvalNgramBin::BlockEval(WordID *wid, int order, float *p)
{
  req.push_back(new NgramReq(wid, order, p));
  if (req.size()>=max_req) BlockFinish();
}

void EvalNgramBin::BlockFinish()
{
#ifdef DEBUG
  for (vector<NgramReq*>::iterator it=req.begin(); it<req.end(); ++it)
    (*it)->display();
#endif
  //sort(req.begin(),req.end());  // use operator < of Ngramreq
  sort(req.begin(),req.begin()+req.size()-1,NgramReq::Compare);
  //qsort(&req[0], req.size(), sizeof(req[0]), NgramReq::Compare);
#ifdef DEBUG
  for (int i=0; i<req.size(); i++) {
    printf("buf %d:", i); req[i]->display();
  }
#endif
  req.clear();
```

```
}
```

D.27 EvalNgramBin.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: EvalNgramBin.h,v 1.3 2010/01/26 19:37:22 schwenk Exp $
 */

#ifndef _EvalNgramBin_h
#define _EvalNgramBin_h

#include <iostream>
#include "Tools.h"
#include "Mach.h"
#include "DataNgramBin.h"

//
// helper class to store and compare one ngram LM request
//
class NgramReq {
  int ctxt_len;
  WordID *ctxt, wpred;
  float *res_ptr;
public:
  NgramReq(WordID *wid, int order, float *adrP)
    : ctxt_len(order-1), ctxt(new WordID[ctxt_len]),
wpred(wid[ctxt_len]), res_ptr(adrP)
    { for (int i=0; i<ctxt_len; i++) ctxt[i]=wid[i]; }
  ~NgramReq() {delete [] ctxt; }
   static bool Compare(NgramReq* n1, NgramReq *n2)
   { return true;
     for (int i=0; i<n1->ctxt_len; i++) {
```

```
      if (n1->ctxt[i] < n2->ctxt[i]) return true;
      if (n1->ctxt[i] > n2->ctxt[i]) return false;
    }
    return true; // both are equal
  }
  void display() {
    for (int c=0; c<ctxt_len; c++) printf(" %d", ctxt[c]);
    printf(" -> %d\n", wpred);
  }
};




class EvalNgramBin
{
private:
protected:
  Mach  &mach;                // network to evaluate
  int idim, odim, bsize;      // copied here for faster access
    // buffer to store bsize examples
  REAL  *buf_input;
    // buffers for block operations
  vector<NgramReq*> req;
  uint      max_req;    // max number of request cumulated before we
perform them in a block
public:
  EvalNgramBin(Mach&, uint=128);          // spezify one machine
  //EvalNgramBin(string, int=16384);            // spezify multiple
ipol machines
  virtual ~EvalNgramBin();
  //virtual void Data(Data &data, int* = NULL); // evaluate on existing
data
  virtual REAL Eval(WordID*, int, float* = NULL);     // get prob for 1
n-gram only
  virtual void BlockEval(WordID*, int, float*);
  virtual void BlockFinish();
};

#endif
```

D.28 Hypo.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
```

```
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Hypo.h,v 1.6 2010/01/25 12:27:07 schwenk Exp $
 *
 * Basic functions to process one hypothesis
 */


#ifndef _HYPO_H_
#define _HYPO_H_

using namespace std;

#include <iostream>
#include <fstream>
#include <string>
#include <vector>

#include "Toolsgz.h"

#define NBEST_DELIM "|||"
#define NBEST_DELIM2 " ||| "

class Hypo {
protected:
  int id;
  string trg;  // translation
  vector<float> f;  // feature function scores
  float     s;  // global score
  // segmentation
public:
  Hypo();
  Hypo(int p_id,string &p_trg, vector<float> &p_f, float p_s) :
id(p_id),trg(p_trg),f(p_f),s(p_s) {};
  ~Hypo();
  float CalcGlobal(Weights&);
  void AddID(int o) {id+=o;};
  void Write(outputfilestream&);
  bool operator< (const Hypo&) const;
  // bool CompareLikelihoods (const Hypo&, const Hypo&) const;
  void SetFeature(float val, const int pos) {if(pos>0) f[pos-1]=val;
else f.push_back(val); };
  const char *GetCstr() {return trg.c_str(); };
};

#endif
```

D.29 Mach.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Mach.cpp,v 1.13 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "Mach.h"
#include "MachTab.h"
#include "MachTabShared.h"
#include "MachLin.h"
#include "MachSig.h"
#include "MachTanh.h"
#include "MachSoftmax.h"
#include "MachSeq.h"
#include "MachPar.h"
#include "cuda_runtime.h"
#include "npps.h"

void Mach::do_alloc()
{
  if (odim*bsize>0) {
    //data_out=::new REAL[odim*bsize];
      cudaMallocManaged(&data_out, odim*bsize*sizeof(REAL));
    if (!data_out) Error ("can't allocate memory for data_out");
  }
  else data_out=NULL;
  cudaMallocManaged(&data_in, sizeof(REAL));
  data_in=NULL; // (luint) this) should be set later by SetDataIn()
  if (idim*bsize>0) {
```

```
    //grad_in=::new REAL[idim*bsize];
    cudaMallocManaged(&grad_in, idim*bsize*sizeof(REAL));
    if (!grad_in) Error ("can't allocate memory for grad_in");
  }
  else grad_in=NULL;
  cudaMallocManaged(&grad_out, sizeof(REAL));
  grad_out=NULL; // (luint) this) should be set later by SetGradOut()
}

Mach::Mach(const int p_idim, const int p_odim, const int p_bsize, const
int p_nbfw, const int p_nbbw)
 : idim(p_idim), odim(p_odim), bsize(p_bsize), nb_forw(p_nbfw),
nb_backw(p_nbbw)
{
  do_alloc();
}

Mach::~Mach()
{
  //if (data_out) delete [] data_out;
  cudaFree(data_out);
  //if (grad_in) delete [] grad_in;
  cudaFree(grad_in);
}

//-----------------------------------------------
// File output
//-----------------------------------------------

void Mach::WriteParams(ofstream &of) {
    // write machine specific params
  of.write((char*) &nb_forw, sizeof(int));
  of.write((char*) &nb_backw, sizeof(int));
}

void Mach::WriteData(ofstream &of) {
  const int i=0, s=sizeof(REAL);
  of.write((char*) &i, sizeof(int));
  of.write((char*) &s, sizeof(int));
}

void Mach::Write(ofstream &of)
{
  char header[file_header_size];
  for (int i=0; i<file_header_size; i++) header[i]=' ';
  sprintf(header,"%s %d",file_header_name, file_header_version);
  of.write(header,file_header_size);
  of.write((char*) &idim, sizeof(int));
  of.write((char*) &odim, sizeof(int));
  of.write((char*) &bsize, sizeof(int));
  int mtype=GetMType();
  of.write((char*) &mtype, sizeof(int));
  WriteParams(of);
  WriteData(of);
}

//-----------------------------------------------
```

```
// File input
//-----------------------------------------------


void Mach::ReadParams(ifstream &inpf, bool with_alloc)
{
  inpf.read((char*) &nb_forw, sizeof(int));
  inpf.read((char*) &nb_backw, sizeof(int));
}

void Mach::ReadData(ifstream &inpf, ptrdiff_t s)
{
  // there is nothing to read
}

Mach *Mach::Read(ifstream &inpf )
{
  char header[file_header_size], h[file_header_size];
  int v;

  inpf.read(header,file_header_size);
  if (sscanf(header,"%s %d",h,&v) != 2) {
    fprintf(stderr,"format of machine file not recognised: %s",
header);
    Error();
  }
  if (strcmp(h,file_header_name)) {
    fprintf(stderr, "unsupported file type (%s), expected '%s'\n", h,
file_header_name);
    Error();
  }
  switch (file_header_version) {
    case file_header_version: break;
    default:
      fprintf(stderr,"unsupported version of machine file (%d)\n",v);
      Error();
  }

    // read idim, odim, bsize
  int f_idim, f_odim, f_bsize;
  inpf.read((char*) &f_idim, sizeof(int));
  inpf.read((char*) &f_odim, sizeof(int));
  inpf.read((char*) &f_bsize, sizeof(int));

   // read and parse machine type
  int mtype;
  Mach *m;
  inpf.read((char*) &mtype, sizeof(int));
  switch (mtype) {
    case file_header_mtype_base: m = new Mach(f_idim,f_odim,f_bsize);
break;
    case file_header_mtype_tab: m = new
MachTab(NULL,f_idim,f_odim,f_bsize,0,0); break;
    case file_header_mtype_lin: m = new MachLin(f_idim,f_odim,f_bsize);
break;
    case file_header_mtype_sig: m = new MachSig(f_idim,f_odim,f_bsize);
break;
```

```
    case file_header_mtype_tanh: m = new
MachTanh(f_idim,f_odim,f_bsize); break;
    case file_header_mtype_softmax: m = new
MachSoftmax(f_idim,f_odim,f_bsize); break;
    case file_header_mtype_multi: m = new MachMulti(); break;
    case file_header_mtype_mseq: m = new MachSeq(); break;
    //case file_header_mtype_mstack: m = new MachStack; break;
    case file_header_mtype_mpar: m = new MachPar(); break;
    default:
      fprintf(stderr,"unknown machine type in file (%d)\n", mtype);
      Error();
  }

    // read rest of (machine specific) params
  m->ReadParams(inpf);

  int s;
  inpf.read((char*) &s,sizeof(int));  // number of elements
  inpf.read((char*) &v,sizeof(int));  // size in bytes of each element
  if (v != sizeof(REAL)) {
    fprintf(stderr, "binary data on file uses %d bytes while the
current code is compiled for %lu bytes\n", v, sizeof(REAL));
    Error();
  }
  m->ReadData(inpf, s);
  // TODO: check EOF

  return m;
}

//-----------------------------------------------
// Tools
//-----------------------------------------------

void Mach::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << " - dimensions: in=" << idim << ", out=" << odim << endl;
    cout << " - number of parallel examples=" << bsize << endl;
    cout << " - number of passes: " << nb_forw << "/" << nb_backw <<
endl;
  }
  else {
    printf("%sMach %d-%d, bs=%d, passes=%d/%d\n", txt, idim, odim,
bsize, nb_forw, nb_backw);
  }
}

//-----------------------------------------------
// Training
//-----------------------------------------------

void Mach::Forw(int eff_bsize)
{
  if (!data_in)
    Error("Mach::Forw(): input data is not set");
  if (idim!=odim)
```

```
    Error("Mach::Forw(): call to default Forw() function with different
dimensions");
  if (eff_bsize<=0) eff_bsize=bsize;
  //memcpy(data_out, data_in, eff_bsize*idim*sizeof(REAL));
  nppsCopy_32f(data_in, data_out, eff_bsize*idim);
  nb_forw += eff_bsize;
}

void Mach::Backw (const float lrate, const float wdecay, int eff_bsize)
{
  if (!grad_out)
    Error("Mach::Backw(): output gradient is not set");
  if (idim!=odim)
    Error("Mach::Backw(): call to default Train() function with
different dimensions");
  if (eff_bsize<=0) eff_bsize=bsize;
  //memcpy(grad_in, grad_out, eff_bsize*idim*sizeof(REAL));
  nppsCopy_32f(grad_out, grad_in, eff_bsize*idim);
  nb_backw += eff_bsize;
}
```

D.30 Mach.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Mach.h,v 1.11 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Machine_h
#define _Machine_h

#include <iostream>
#include <fstream>
```

```
#include "Tools.h"

#define BLAS      // use fast BLAS code, for instance with Intel's MKL
library

// list of all known machine types,
// this is needed for the general file read function

#define file_header_name "HPerf"
#define file_header_version 1
#define file_header_size 16

#define file_header_mtype_base            0
#define file_header_mtype_tab       1
#define file_header_mtype_tabsh           2
#define file_header_mtype_lin       3
#define file_header_mtype_sig       4
#define file_header_mtype_tanh            5
#define file_header_mtype_softmax   6
#define file_header_mtype_stab            7
#define file_header_mtype_multi           16
#define file_header_mtype_mseq            17
#define file_header_mtype_mstack    18
#define file_header_mtype_mpar            19

class Mach
{
private:
  void do_alloc();      // perform allocation of dynamic data
structures
protected:
  int  idim, odim;            // input and output dimension
  int  bsize;                 // block size (nb of example used in
parallel)
  int  nb_forw;               // nb of forward examples processed
  int  nb_backw;        // nb of backward examples processed
  REAL *data_in;        // input data (pointer)
  REAL *data_out;       // output data (allocated by machine)
  REAL *grad_in;        // input gradients (allocated by machine)
  REAL *grad_out;       // output gradients (pointer)
  // File I/O, the following functions can be overloaded by subclass
  // the main functions Read() and Write() should not be modified !
  virtual void ReadParams(ifstream&, bool=true); // read all params
  virtual void ReadData(ifstream&, ptrdiff_t); // read binary data
  virtual void WriteParams(ofstream&); // write all params
  virtual void WriteData(ofstream&); // write binary data
public:
  Mach(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~Mach();
    // Tools
  virtual int GetMType() {return file_header_mtype_base;};  // get type
of machine
  virtual int GetIdim() {return idim;}
  int GetOdim() {return odim;}
  int GetBsize() {return bsize;}
  void SetBsize(int bs) {
```

```
    if (bs<1) Error("wrong value in SetBsize()"); else bsize=bs; }
  int GetNbForw() {return nb_forw;}
  int GetNbBackw() {return nb_backw;}
  virtual REAL* GetDataIn() {return data_in;} // return pointer on
input data for chaining
  virtual REAL* GetDataOut() {return data_out;} // return pointer on
output data for chaining
  virtual REAL* GetGradIn() {return grad_in;} // return pointer on
input gradient for chaining
  virtual REAL* GetGradOut() {return grad_out;} // return pointer on
output gradient for chaining
  virtual void SetDataIn(REAL *data) {data_in=data;} // set pointer of
input data
  virtual void SetGradOut(REAL *data) {grad_out=data;} // set pointer
of output gradient
  virtual void Info(bool=false, char *txt=(char*)" - ");// display
(detailed) information on machine
    // FILE IO
  static Mach *Read(ifstream&);     // read class from a stream
  void Write(ofstream&); // write content of class to a stream
    // Training
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int =0);
};

#endif
```

D.31 MachLin.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachLin.cpp,v 1.17 2010/01/26 11:05:27 schwenk Exp $
```

```
 */

using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();
#include "my_cuda.h"
#include "Tools.h"
#include "MachLin.h"
#include "Blas.h"
#include "Gpu.cuh"

REAL *d_C;// = 0;
MachLin::MachLin(const int p_idim, const int p_odim, const int p_bsize,
const int p_nbfw, const int p_nbbw)
 : Mach(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
  if (odim>0) {
    //b = new REAL[odim];
        cudaMallocManaged(&b, odim*sizeof(REAL));
    if (!b) Error ("can't allocate memory for bias of linear machine");
  }
  else b=NULL;
  if (idim*odim>0) {
    //w = new REAL[idim*odim];
      cudaMallocManaged(&w, idim*odim*sizeof(REAL));
      if (!w) Error ("can't allocate memory for weights of linear
machine");
  }
  else w=NULL;
}

MachLin::~MachLin()
{

#if 0
  printf("W:\n");
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
  printf("b: ");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
  //if (b) delete [] b;
  cudaFree(b);
  //if (w) delete [] w;
  cudaFree(w);
}

void MachLin::BiasConst(const REAL val)
{
  for (int i=0; i<odim; i++) b[i]=val;
}

void MachLin::BiasRandom(const REAL range)
```

```
{
  REAL c=range*2.0;
  for (int i=0; i<odim; i++) b[i]=c*(drand48()-0.5);
}

void MachLin::WeightsConst(const REAL val)
{
  for (int i=0; i<idim*odim; i++) w[i]=val;
}

void MachLin::WeightsRandom(const REAL range)
{
  REAL c=range*2.0;
  for (int i=0; i<idim*odim; i++) w[i]=c*(drand48()-0.5);
}

void MachLin::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on linear machine" << endl;
    Mach::Info(detailed,txt);
  }
  else {
    printf("%sMachLin %d-%d, bs=%d, passes=%d/%d\n", txt, idim, odim,
bsize, nb_forw, nb_backw);
  }
}

//-----------------------------------------------
// File output
//-----------------------------------------------

void MachLin::WriteData(ofstream &outf) {
  int s=odim*idim + odim;
  outf.write((char*) &s,sizeof(int));
  s=sizeof(REAL);
  outf.write((char*) &s,sizeof(int));
  outf.write((char*) w,odim*idim*sizeof(REAL));
  outf.write((char*) b,odim*sizeof(REAL));
#if 0
cout << "\nWriting on file:" << endl;
  printf("W: %dx%d\n",odim,idim);
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
  printf("b:\n");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
}

//-----------------------------------------------
// File input
//-----------------------------------------------
```

```
void MachLin::ReadData(ifstream &inpf, ptrdiff_t s)
{
  ptrdiff_t se=odim*idim + odim;
  if (s!=se) {
    cerr << "ERROR: data block of linear machine has " << s << "
elements (" << se << " were expected)" << endl;    Error();
  }
  Mach::ReadData(inpf, 0);
    // read parameters
    // TODO: error checks
  inpf.read((char*) w,odim*idim*sizeof(REAL));
  inpf.read((char*) b,odim*sizeof(REAL));
#if 0
cout << "\nRead from file:" << endl;
  printf("W: %dx%d\n",odim,idim);
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
  printf("b:\n");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
}


//-----------------------------------------------
// Training
//-----------------------------------------------

void MachLin::Forw(int eff_bsize)
{
  //printf("entering MachLin Forw.... \n");
  if (!data_in)
    Error("MachLin::Forw(): input data is not set");

  if (eff_bsize<=0) eff_bsize=bsize;

#if 0
  printf("Forw %p, bsize=%d\n", (void*)this, eff_bsize);
  printf("W: %dx%d\n",odim,idim);
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
  printf("b:\n");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
#if 0
for (int e=0; e<eff_bsize; e++) {
  printf("B %d inp:", e);
  for (int i=0; i<idim; i++) printf(" %7.5f", data_in[i+e*idim]);
  printf("\n");
}
#endif
```

```
#ifdef BLAS
  if (eff_bsize>1)
  { // BLAS block mode: GEMM
    int e,o;
    float alpha = 1.0f;
    float beta = 1.0f;
    REAL *optr, *bptr;

//    REAL *d_A = 0;
//    REAL *d_B = 0;
//    REAL *d_C = 0;

 //   for (int e=0; e<eff_bsize; e++)
 //       nppsCopy_32f(b,data_out+e*odim,odim);

    // copy bias <eff_bsize> times into result matrix
    GpuCopyVectorToMatrix(data_out, b, eff_bsize, odim);

//    int n2A = odim * idim;
//    int n2B = idim * eff_bsize;
//    int n2C = odim * eff_bsize;

    cublas_stat = cublasSgemm(cublas_handle, CUBLAS_OP_N, CUBLAS_OP_N,
odim, eff_bsize, idim, &alpha, w, odim, data_in, idim, &beta, data_out,
odim);
    if (cublas_stat != CUBLAS_STATUS_SUCCESS)
    {
      fprintf(stderr, "!!!! kernel execution error.\n");
        Error();
    }
  }
  else
  { // BLAS vector mode: GEMV
    call_gemv (data_out, w, data_in, b, odim, idim);
  }
#else
  for (int e=0; e<eff_bsize; e++) {
      // simple matrix vector multiply, TODO: verify bsize
      // TODO: W is stored in BLAS (Fortan) format: colum major !!
    //cout << "forw ex " << e << endl;
    REAL *wptr=w;
    for (int o=0; o<odim; o++) {
      REAL s=b[o];
      for (int i=0; i<idim; i++) s+=wptr[i*odim+o]*data_in[i+e*idim];
      data_out[o+e*odim]=s;
    }
  }
#endif
  nb_forw += eff_bsize;

#if 0
for (int e=0; e<eff_bsize; e++) {
  printf("B %d out:", e);
  for (int i=0; i<odim; i++) printf(" %7.5f", data_out[i+e*odim]);
  printf("\n\n");
}
#endif
```

```
}


void MachLin::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  static REAL real1=1.0, real0=0.0;
  //static char transN='N', transT='T';
  REAL epsilon = 1.0 + lrate * wdecay;

  if (eff_bsize<=0) eff_bsize=bsize;
  if (!grad_out)
    Error("MachLin::Backw(): output gradient is not set");

#if 0
for (int e=0; e<eff_bsize; e++) {
  printf(" B %d grad:", e);
  for (int i=0; i<idim; i++) printf(" %7.5f", grad_out[i]);
  printf("\n");
}
#endif

    // update bias vector:   b = b + lrate * grad_out
    // NO weight decay
  REAL *gptr = grad_out;
//  for (int e=0; e<eff_bsize; e++) {
//    REAL *aptr = b;
//    for (int i=0; i<odim; i++) *aptr++ += lrate * *gptr++;
//  }

  // update bias vector:   b = b + lrate * grad_out
  // NO weight decay
//REAL *gptr = grad_out;
//for (int e=0; e<eff_bsize; e++, gptr+=odim) {
  //AXPY(&odim,&lrate,gptr,&inc1,b,&inc1);
      GpuBatchedAXPY(odim,lrate,grad_out,1,b,1,eff_bsize);
//}

//  for (int e=0; e<eff_bsize; e++, gptr+=odim)
//  {
//    cublas_stat = cublasSaxpy(cublas_handle,odim,&lrate,gptr,1,b,1);
//    if (cublas_stat != CUBLAS_STATUS_SUCCESS)
//    {
//          fprintf(stderr, "!!!! cublasSaxpy kernel execution
error.\n");
//          Error();
//    }
//  }

#if 0
  printf("b after update:\n");
  for (int od=0;od<odim;od++) printf(" %9.7f",b[od]);
  printf("\n");
#endif
  int n2A = idim * odim;
  int n2B = odim * eff_bsize;
```

```
//  int n2Cb = idim * eff_bsize;
//  REAL *d_A = 0;
//  REAL *d_B = 0;
//  REAL *d_Cb = 0;
  /* Allocate device memory for the matrices */
//  if (cudaMalloc((void **)&d_B, n2B * sizeof(d_B[0])) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! device memory allocation error (allocate
B)\n");
//      Error();
//  }
  /* Initialize the device matrices with the host matrices */
//  cublas_stat =
cublasSetMatrix(odim,eff_bsize,sizeof(grad_out[0]),grad_out,odim,d_B,od
im);
//  if (cublas_stat != CUBLAS_STATUS_SUCCESS)
//  {
//      fprintf(stderr, "!!!! device access error (write grad_out)\n");
//      Error();
//  }
    // backprop gradient:   grad_in   =        w'        *   grad_out
    //                    idim x bsize = (odim x idim)'  *  odim x
bsize
//printf("GEMM(%lx=%lx * % x)\n",grad_in, w, grad_out);
  //GEMM (CblasColMajor,CblasTrans, CblasNoTrans, &idim, &eff_bsize,
&odim,
  //      &real1, w, &odim, grad_out, &odim,
  //      &real0, grad_in, &idim);
  //m=idim
  //n=eff_bsize
  //k=odim
  //A=w
  //lda=odim
  //B=grad_out
  //ldb=odim
  //C=grad_in
  //ldc=idim
//  printf("GEMM Dimensions m=%i, n=%i, k=%i \n", idim, eff_bsize,
odim);
//  GEMM (CblasColMajor,CblasTrans, CblasNoTrans, idim, eff_bsize,
odim,real1, w, odim, grad_out, odim, real0, grad_in, idim);
  /* Performs operation using cublas */
//  cublas_stat = cublasSgemm(cublas_handle, CUBLAS_OP_T, CUBLAS_OP_N,
idim, eff_bsize, odim, &real1, w, odim, d_B, odim, &real0, grad_in,
idim);
//  if (cublas_stat != CUBLAS_STATUS_SUCCESS)
//  {
//      fprintf(stderr, "!!!! kernel execution error.\n");
//      Error();
//  }
  cublas_stat = cublasSgemm(cublas_handle, CUBLAS_OP_T, CUBLAS_OP_N,
idim, eff_bsize, odim, &real1, w, odim, grad_out, odim, &real0,
grad_in, idim);
  if (cublas_stat != CUBLAS_STATUS_SUCCESS)
  {
        fprintf(stderr, "!!!! kernel execution error.\n");
        Error();
```

```
  }
    // update weights including weight decay
    // w = lrate  *grad_out * data_in^T + epsilon * w
    // gemm (transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c,
ldc )
    //                                        Go      Din            W
    //        C = alpha*A * B + beta * b
    //

#if 0
  printf("W before update:\n");
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
#endif
//printf("GEMM(%lx=%lx * % x)\n",w, grad_out, data_in);
  //GEMM (CblasColMajor,CblasNoTrans,CblasTrans, &odim, &idim,
&eff_bsize,
  //      &lrate, grad_out, &odim, data_in, &idim,
  //      &epsilon, w, &odim);

//  n2A = eff_bsize * idim;
//  n2Cb = idim * odim;
  /* Allocate device memory for the matrices */
//  if (cudaMalloc((void **)&d_A, n2A * sizeof(d_A[0])) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! device memory allocation error (allocate
A)\n");
//      Error();
//  }
//  cublas_stat =
cublasSetMatrix(idim,eff_bsize,sizeof(data_in[0]),data_in,idim,d_A,idim
);//added to reduce memcopys
//  if (cublas_stat != CUBLAS_STATUS_SUCCESS)
//  {
//      fprintf(stderr, "!!!! device access error (write grad_out)\n");
//      Error();
//  }
//  GEMM (CblasColMajor,CblasNoTrans,CblasTrans, odim, idim,
eff_bsize,lrate, grad_out, odim, data_in, idim, epsilon, w, odim);
//  cublas_stat = cublasSgemm(cublas_handle, CUBLAS_OP_N, CUBLAS_OP_T,
odim, idim, eff_bsize, &lrate, d_B, odim, d_A, idim, &epsilon, w,
odim);

  cublas_stat = cublasSgemm(cublas_handle, CUBLAS_OP_N, CUBLAS_OP_T,
odim, idim, eff_bsize, &lrate, grad_out, odim, data_in, idim, &epsilon,
w, odim);
  if (cublas_stat != CUBLAS_STATUS_SUCCESS)
  {
        fprintf(stderr, "!!!! kernel execution error.\n");
        Error();
  }
//  if (cudaFree(d_A) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! memory free error backw(d_A)\n");
//      Error();
```

```
//  }
//  if (cudaFree(d_B) != cudaSuccess)
//  {
//      fprintf(stderr, "!!!! memory free error backw(d_B)\n");
//      Error();
//  }
//  cudaDeviceSynchronize();
  /* Shutdown */
  //cublasDestroy(handle);

#if 0
  printf("W after update:\n");
  for (int od=0;od<odim;od++) {
    for (int id=0;id<idim;id++) printf(" %9.7f",w[id*odim+od]);
    printf("\n");
  }
#endif
  nb_backw += eff_bsize;
}

void MachLin::Debug()
{
  for (int o=0; o<odim; o++) {
    for (int i=0; i<idim; i++) {
      w[i*odim+o] = i + 1000*o;
    }
    b[o] = -o;
  }
}
```

D.32 MachLin.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

```
 *
 * $Id: MachLin.h,v 1.12 2010/01/25 12:27:07 schwenk Exp $
 *
 * linear machine:  output = weights * input + biases
 */

#ifndef _MachLin_h
#define _MachLin_h

#include "Mach.h"

class MachLin : public Mach
{
protected:
  REAL *b;        // biases
  REAL *w;        // weights, stored in BLAS format, e.g. COLUMN major
!
  virtual void ReadData(ifstream&, ptrdiff_t); // read binary data
  virtual void WriteData(ofstream&); // write binary data
public:
  MachLin(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~MachLin();
  virtual int GetMType() {return file_header_mtype_lin;};   // get type
of machine
  virtual void BiasConst(const REAL val); // init biases with constant
values
  virtual void BiasRandom(const REAL range);    // random init of
biases in [-range, range]
  virtual void WeightsConst(const REAL val);    // init weights with
constant values
  virtual void WeightsRandom(const REAL range); // random init of
weights in [-range, range]
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
  virtual void Debug ();
};

#endif
```

D.33 MachMulti.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
```

```
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachMulti.cpp,v 1.14 2010/01/28 09:27:12 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "MachMulti.h"

MachMulti::MachMulti()
 : Mach(0,0,0)
{
  machs.clear();
}

MachMulti::~MachMulti()
{
  machs.clear();
}

void MachMulti::Delete()
{
  for (unsigned int m=0; m<machs.size(); m++) delete machs[m];
}

void MachMulti::MachAdd(Mach *new_mach)
{
  Error("MachAdd not defined for abstract multiple machine");
}

Mach *MachMulti::MachDel()
{
  Error("MachDel not defined for abstract multiple machine");
  return NULL;
}

//-----------------------------------------------
// File output
//-----------------------------------------------


void MachMulti::WriteParams(ofstream &of) {
  Mach::WriteParams(of);
```

```
  int nbm=machs.size();
  of.write((char*) &nbm, sizeof(int));
}

void MachMulti::WriteData(ofstream &outf) {
  int nbm=machs.size(), s=sizeof(REAL);
  outf.write((char*) &nbm, sizeof(int));
  outf.write((char*) &s, sizeof(int));
  for (vector<Mach*>::iterator it = machs.begin(); it!=machs.end();
++it) {
    (*it)->Write(outf);
  }
}

//-----------------------------------------------
// File input
//-----------------------------------------------

void MachMulti::ReadParams(ifstream &inpf, bool with_alloc)
{
  if (machs.size() > 0)
    Error("Trying to read multiple machine into non empty data
structures\n");

  Mach::ReadParams(inpf, false);
  int nbm;
  inpf.read((char*) &nbm, sizeof(int));
  if (nbm<1) Error("illegal number of machines");
  machs.clear();
  for (int i=0; i<nbm; i++) machs.push_back(NULL);
}

void MachMulti::ReadData(ifstream &inpf, ptrdiff_t s)
{
  if (s!=machs.size()) {
    cerr << "ERROR: data block of multiple machine has " << s << "
machines (" << machs.size() << " were expected)" << endl;    Error();
  }

  for (vector<Mach*>::iterator it = machs.begin(); it!=machs.end();
++it) {
    (*it) = Mach::Read(inpf);
  }
}

//
// Tools
//

void MachMulti::SetBsize(int bs)
{
  if (bs<1) Error("wrong value in SetBsize()");
  for (uint i=0; i<machs.size(); i++) machs[i]->SetBsize(bs);
}

void MachMulti::Info(bool detailed, char *txt)
{
```

```
  if (detailed) {
    if (machs.size()) {
      Mach::Info();
      for (unsigned int i=0; i<machs.size(); i++) {
        cout << "MACHINE " << i << ": " << endl;
        machs[i]->Info();
      }
    }
    else
      cout << " *** empty ***" << endl;
  }
  else {
    printf("%sMultiple machine %d- .. -%d, bs=%d, passes=%d/%d\n", txt,
idim, odim, bsize, nb_forw, nb_backw);
    char ntxt[256];
    sprintf(ntxt,"%s  ", txt);
    for (unsigned int i=0; i<machs.size(); i++) machs[i]-
>Info(detailed, ntxt);
  }
}

void MachMulti::Forw(int eff_bszie)
{
  if (machs.empty())
    Error("called Forw() for an empty multiple machine");
  else
    Error("call to Forw() not defined for an abstract multiple
machine");
}

void MachMulti::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  Error("call to Backw() not defined for an abstract multiple
machine");
}
```

D.34 MachMulti.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
```

```
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachMulti.h,v 1.10 2010/01/26 19:37:22 schwenk Exp $
 *
 * virtual class to support various combinations of multiple machines
 */

#ifndef _MachMulti_h
#define _MachMulti_h

using namespace std;
#include <vector>

#include "Mach.h"

class MachMulti : public Mach
{
protected:
  vector<Mach*> machs;
  virtual void ReadParams(ifstream&, bool =true);
  virtual void ReadData(ifstream&, ptrdiff_t); // read binary data
  virtual void WriteParams(ofstream&); // write all params
  virtual void WriteData(ofstream&); // write binary data
public:
  MachMulti();    // create initial sequence with no machine
  virtual ~MachMulti();
  virtual int GetMType() {return file_header_mtype_multi;}; // get type
of machine
  void SetBsize(int bs);
    // add and remove machines
  virtual void Delete();      // call destructor for all the machines
  virtual void MachAdd(Mach*);      // add new machine after the
existing ones
  virtual Mach *MachDel();    // delete the last machine
    // standard functions
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);  // calculate outputs for current inputs
  virtual void Backw(const float lrate, const float wdecay, int=0); //
calculate gradients at input for current gradients at output
};

#endif
```

D.35 MachPar.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
```

```
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachPar.cpp,v 1.12 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"
#include "MachTab.h"
#include "MachPar.h"
#include "npps.h"
#include "cuda_runtime.h"

void MachPar::do_alloc()
{
  //if (data_out) delete [] data_out;
  //if (grad_in) delete [] grad_in;
  //data_out = (odim*bsize>0) ? new REAL[odim*bsize] : NULL;
      cudaMallocManaged(&data_out, odim*bsize*sizeof(REAL));
  //grad_in = (idim*bsize>0) ? new REAL[idim*bsize] : NULL;
      cudaMallocManaged(&grad_in, idim*odim*sizeof(REAL));
}


MachPar::MachPar()
 : MachMulti()
{
}

MachPar::~MachPar()
{
  // data_out and grad_in will be freed by Mach::~Mach()
}

void MachPar::MachAdd(Mach *new_mach)
{
  if (machs.empty()) {
```

```
    machs.push_back(new_mach);
      // think about freeing memory
    idim=new_mach->GetIdim();
    odim=new_mach->GetOdim();
    bsize=new_mach->GetBsize();
    data_in=NULL; // will be set by MachPar::SetDataIn()
    data_out=NULL;
    grad_in = NULL;
    grad_out = NULL;
    do_alloc();
    new_mach->SetGradOut(grad_out);
  }
  else {
    if (bsize!=new_mach->GetBsize())
      Error("bunch size of new parallel machine does not match");
    machs.push_back(new_mach);

      // resize input gradient and output data
    idim += new_mach->GetIdim();
    odim += new_mach->GetOdim();
    do_alloc();
  }
}

Mach *MachPar::MachDel()
{
  if (machs.empty()) {
    Error("impossible to delete element from parallel machine: is
already empty");
  }

  Error("TODO");
  return NULL;
}

// set pointer of input data
void MachPar::SetDataIn(REAL *data)
{
  data_in=data;
    // set input data of indiv machines one after each other
    // this depends on the effective bsize !
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->SetDataIn(data);
    data += bsize*machs[m]->GetIdim();
  }
}

// set pointer of output gradient
void MachPar::SetGradOut(REAL *data)
{
  grad_out=data;
    // set output gradients of indiv machines one after each other
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->SetGradOut(data);
    data += bsize*machs[m]->GetOdim();
  }
}
```

```
//---------------------------------------------
// File output
//---------------------------------------------

void MachPar::ReadData(ifstream &inpf, ptrdiff_t s)
{
  MachMulti::ReadData(inpf,s);

     // calculate idim and odim and and allocate data_out and grad_in
  idim=odim=0;
  for (uint m=0; m<machs.size(); m++) {
    idim += machs[m]->GetIdim();
    odim += machs[m]->GetOdim();
  }
  bsize = machs[0]->GetBsize();
  do_alloc();

    // scanning for MachTab with shared addresses
  REAL *tadr=NULL;
  for (uint m=0; m<machs.size(); m++) {
    MachTab *mt= (MachTab*) machs[m];
    if (mt->GetMType()==file_header_mtype_tab) {
      if (mt->GetTabAdr()) {
        if (tadr) {
        }
        else {
        }
        tadr=mt->GetTabAdr();
      }
      else {
        mt->SetTabAdr(tadr);
      }
    }
  }
}

//
// Tools
//

void MachPar::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on parallel machine" << endl;
    MachMulti::Info(detailed);
  }
  else {
    printf("%sParallel machine %d- .. %d, bs=%d, passes=%d/%d\n", txt,
idim, odim, bsize, nb_forw, nb_backw);
    char ntxt[256];
    sprintf(ntxt,"%s  ", txt);
    for (unsigned int i=0; i<machs.size(); i++) machs[i]-
>Info(detailed, ntxt);
  }
}
```

```
// forward pass for all machines and copy output into cumulated output
void MachPar::Forw(int eff_bsize)
{
  if (machs.empty())
    Error("called Forw() for an empty parallel machine");

  if (eff_bsize<=0) eff_bsize=bsize;

      // we need to set the pointers to the input data of indiv
machines
      // one after each other since this depends on the effective bsize
!
//  printf("Entering MachPar Forw... \n");
//  cudaDeviceSynchronize();
  REAL *iptr=data_in;
  REAL *optr=data_out;
//  printf("MachPar Forw pointers ok... \n");
  for (unsigned int m=0; m<machs.size(); m++)
  {
    machs[m]->SetDataIn(iptr);
//    printf("MachPar Forw SetDataIn(iptr)... \n");
    machs[m]->Forw(eff_bsize);
//    printf("Starting MachPar memcpy...\n");
    //memcpy(optr, machs[m]->GetDataOut(), eff_bsize*machs[m]-
>GetOdim()*sizeof(REAL));
    nppsCopy_32f(machs[m]->GetDataOut(),optr,eff_bsize*machs[m]-
>GetOdim());
    iptr += eff_bsize*machs[m]->GetIdim();
    optr += eff_bsize*machs[m]->GetOdim();
  }
  nb_forw += eff_bsize;
  //printf("par forw ok!!!! \n");
}

// backward pass for all machines and copy input gradient into
cumulated gradient
void MachPar::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  if (machs.empty())
    Error("called Backw() for an empty parallel machine");
  if (eff_bsize<=0) eff_bsize=bsize;

      // we need to set the pointers to output gradients of indiv
machines
      // one after each other since this depends on the effective bsize
!
//  printf("Entering MachPar Backw... \n");
  REAL *gptr=grad_in;
  REAL *optr=grad_out;
  for (unsigned int m=0; m<machs.size(); m++) {
    machs[m]->SetGradOut(optr);
    machs[m]->Backw(lrate,wdecay,eff_bsize);
    //memcpy(gptr, machs[m]->GetGradIn(), eff_bsize*machs[m]-
>GetIdim()*sizeof(REAL));
```

```
    nppsCopy_32f(machs[m]->GetGradIn(),gptr,eff_bsize*machs[m]-
>GetIdim());
    optr += eff_bsize*machs[m]->GetOdim();
    gptr += eff_bsize*machs[m]->GetIdim();
  }
  cudaDeviceSynchronize();
  nb_backw += eff_bsize;
}
```

D.36 MachPar.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachPar.h,v 1.9 2010/01/25 12:27:07 schwenk Exp $
 *
 *  Parallel machine:
 *   - put several machine in parallel with a concatenated input and
output layer
 *   - the dimensions of the input and output layers may be different
 */

#ifndef _MachPar_h
#define _MachPar_h

using namespace std;
#include <vector>

#include "MachMulti.h"

class MachPar : public MachMulti
{
private:
```

```
  void do_alloc();      // perform allocation of dynamic data
structures
protected:
  virtual void ReadData(ifstream&, ptrdiff_t); // read binary data
public:
  MachPar();      // create initial sequence with no machine
  virtual ~MachPar();
  virtual int GetMType() {return file_header_mtype_mpar;};  // get type
of machine
    // redfine connecting functions
  virtual void SetDataIn(REAL*);    // set pointer of input data
  virtual void SetGradOut(REAL*);   // set pointer of output gradient
    // add and remove machines
  virtual void MachAdd(Mach*); // add new machine after the existing
ones
  virtual Mach *MachDel();
    // standard functions
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
  virtual void Backw(const float lrate, const float wdecay, int=0);
      // calculate gradients at input for current gradients at output
};

#endif
```

D.37 MachSeq.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSeq.cpp,v 1.14 2010/01/26 11:05:27 schwenk Exp $
 */
```

```
using namespace std;
#include <iostream>

#include "Tools.h"
#include "MachSeq.h"
//#include "cuda_runtime.h"

MachSeq::MachSeq()
 : MachMulti()
{
}

MachSeq::~MachSeq()
{
  data_out=grad_in=NULL;  // prevent delete[] by ~Mach()
}

// set pointer of input data
void MachSeq::SetDataIn(REAL *data)
{
  data_in=data;
  if (machs.size() > 0) machs[0]->SetDataIn(data);
}

// set pointer of output gradient
void MachSeq::SetGradOut(REAL *data)
{
  grad_out=data;
  if (machs.size() > 0) machs.back()->SetGradOut(data);
}

void MachSeq::MachAdd(Mach *new_mach)
{
  if (machs.empty()) {
    machs.push_back(new_mach);
      // think about freeing memory
    idim=new_mach->GetIdim();
    bsize=new_mach->GetBsize();
    data_in=new_mach->GetDataIn();
    grad_in=new_mach->GetGradIn();
  }
  else {
    Mach *last_mach=machs.back();
    if (last_mach->GetOdim()!=new_mach->GetIdim()) {
      cout << "Current sequential machine:" << endl; Info(false);
      cout << "Newly added machine:" << endl; new_mach->Info(false);
      Error("input dimension of new sequential machine does not
match");
    }
    if (bsize!=new_mach->GetBsize()) {
      cout << "Current sequential machine:" << endl; Info(false);
      cout << "Newly added machine:" << endl; new_mach->Info(false);
      Error("bunch size of new sequential machine does not match");
    }
    machs.push_back(new_mach);

      // connect new last machine to the previous one
```

```
    new_mach->SetDataIn(last_mach->GetDataOut());
    last_mach->SetGradOut(new_mach->GetGradIn());
  }

    // connect last machine to the outside world
  odim=new_mach->GetOdim();
  data_out=new_mach->GetDataOut();
  grad_out=new_mach->GetGradOut();
}

Mach *MachSeq::MachDel()
{
  if (machs.empty()) {
    Error("impossible to delete element from sequential machine: is
already empty");
  }

  Mach *del_mach=machs.back();
  machs.pop_back();

  if (machs.empty()) {
    idim=odim=bsize=0;
    data_in=data_out=grad_in=grad_out=NULL;
  }
  else {
    Mach *last_mach=machs.back();

      // connect new last machine to the outside world
    odim=last_mach->GetOdim();
    data_out=last_mach->GetDataOut();
    grad_out=last_mach->GetGradOut();
  }

  return del_mach;
}

//-----------------------------------------------
// File input
//-----------------------------------------------

void MachSeq::ReadData(ifstream &inpf, size_t s)
{
  MachMulti::ReadData(inpf,s);


  int nbm=machs.size();
  idim = machs[0]->GetIdim();
  bsize = machs[0]->GetBsize();
  odim = machs[nbm-1]->GetOdim();

    // connect first to the outside world
  data_in=machs[0]->GetDataIn();
  grad_in=machs[0]->GetGradIn();

    // forward chain the data
  for (int m=1; m<nbm; m++) machs[m]->SetDataIn(machs[m-1]-
>GetDataOut());
```

```
    // backward chain the gradients
  for (int m=nbm-1; m>0; m--) machs[m-1]->SetGradOut(machs[m]-
>GetGradIn());

    // connect last machine to the outside world
  data_out=machs[nbm-1]->GetDataOut();
  grad_out=machs[nbm-1]->GetGradOut();
}

//
// Tools
//

void MachSeq::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on stacked machine" << endl;
    MachMulti::Info(detailed,txt);
  }
  else {
    printf("%sSequential machine [%u] %d- .. -%d, bs=%d,
passes=%d/%d\n", txt, (uint) machs.size(), idim, odim, bsize, nb_forw,
nb_backw);
    char ntxt[256];
    sprintf(ntxt,"%s  ", txt);
    for (unsigned int i=0; i<machs.size(); i++) machs[i]-
>Info(detailed, ntxt);
  }
}

void MachSeq::Forw(int eff_bsize)
{
//  printf("Entering MachSeq Forw... \n");
  if (machs.empty())
    Error("called Forw() for an empty sequential machine");
  for (unsigned int i=0; i<machs.size(); i++)
  {
        //printf("Next Machine!!! \n");
        machs[i]->Forw(eff_bsize);
  }
  nb_forw += (eff_bsize<=0) ? bsize : eff_bsize;
  //printf("MachSeq ok!!! \n");
}

void MachSeq::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
//    printf("Entering MachSeq Back... \n");
      if (machs.empty())
    Error("called Backw() for an empty sequantial machine");
  for (int i=machs.size()-1; i>=0; i--) {
       machs[i]->Backw(lrate,wdecay,eff_bsize);
  }
  nb_backw += (eff_bsize<=0) ? bsize : eff_bsize;
 // cudaDeviceSynchronize();
}
```

D.38 MachSeq.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSeq.h,v 1.8 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _MachSeq_h
#define _MachSeq_h

using namespace std;
#include <vector>

#include "MachMulti.h"

class MachSeq : public MachMulti
{
protected:
  virtual void ReadData(ifstream&, size_t); // read binary data
public:
  MachSeq();      // create initial sequence with no machine
  virtual ~MachSeq();
  virtual int GetMType() {return file_header_mtype_mseq;};  // get type
of machine
    // redfine connecting functions
  virtual void SetDataIn(REAL*);    // set pointer of input data
  virtual void SetGradOut(REAL*);   // set pointer of output gradient
    // add and remove machines
  virtual void MachAdd(Mach*); // add new machine after the existing
ones
  virtual Mach *MachDel();
    // standard functions
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
```

```
  virtual void Forw(int=0);   // calculate outputs for current inputs
  virtual void Backw(const float lrate, const float wdecay, int=0);
      // calculate gradients at input for current gradients at output
};

#endif
```

D.39 MachSig.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSig.cpp,v 1.10 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <math.h>
//extern double drand48();

#include "Tools.h"
#include "MachSig.h"

MachSig::MachSig(const int p_idim, const int p_odim, const int p_bsize,
const int p_nbfw, const int p_nbbw)
 : MachLin(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
}

MachSig::~MachSig()
{
  printf("** destructor MachSig %lx\n",(luint) this);
}
```

```
//---------------------------------------------
// Tools
//---------------------------------------------

void MachSig::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on sigmoidal machine" << endl;
    MachLin::Info(detailed,txt);
  }
  else {
    printf("%sMachSig %d-%d, bs=%d, passes=%d/%d\n", txt, idim, odim,
bsize, nb_forw, nb_backw);
  }
}

//---------------------------------------------
// Training
//---------------------------------------------

void MachSig::Forw(int eff_bsize)
{
//  printf("entering MachSig Forw.... \n");
  if (eff_bsize<=0) eff_bsize=bsize;
  MachLin::Forw(eff_bsize);

    // apply sigmoid on output
#ifdef BLAS
  Error("implement sigmoid\n");
#else
  Error("implement sigmoid\n");
#endif
}

void MachSig::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
    // derivate sigmoidal activation function
    //             = grad_hidden .* ( 1 - a_hidden^2 )

  REAL *aptr = data_out;
  REAL *gptr = grad_out;

  if (eff_bsize<=0) eff_bsize=bsize;
  if (!grad_out)
    Error("MachSig::Backw(): output gradient is not set");

  for (int i=0; i<odim*eff_bsize; i++) {
    REAL val = *aptr++;
    Error("implement derivative of sigmoid\n");
    *gptr=val;
  }

  MachLin::Backw(lrate, wdecay, eff_bsize);
}
```

D.40 MachSig.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSig.h,v 1.7 2010/01/25 12:27:07 schwenk Exp $
 *
 * sigmoidal machine:  output = tanh(weights * input + biases)
 */

#ifndef _MachSig_h
#define _MachSig_h

#include "MachLin.h"

class MachSig : public MachLin
{
public:
  MachSig(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~MachSig();
  virtual int GetMType() {return file_header_mtype_sig;};   // get type
of machine
  virtual void Info(bool=false, char *txt= (char*)"");      // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
};

#endif
```

D.41 MachSoftmax.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSoftmax.cpp,v 1.16 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include "math.h"
#include "my_cuda.h"
#include "Tools.h"
#include "MachSoftmax.h"
#include "Blas.h"
#include "Gpu.cuh"

//static REAL *gpu_result;
REAL *d_softmax=0;
MachSoftmax::MachSoftmax(const int p_idim, const int p_odim, const int
p_bsize, const int p_nbfw, const int p_nbbw)
 : MachLin(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{

}

MachSoftmax::~MachSoftmax()
{
}

//-----------------------------------------------
// Tools
//-----------------------------------------------

void MachSoftmax::Info(bool detailed, char *txt)
```

```
{
  if (detailed) {
    cout << "Information on softmax machine" << endl;
    MachLin::Info(detailed);
  }
  else {
    printf("%sMachSoftmax %d-%d, bs=%d, passes=%d/%d\n", txt,idim,
odim, bsize, nb_forw, nb_backw);
  }
}

//-------------------------------------------------
// Training
//-------------------------------------------------

void MachSoftmax::Forw(int eff_bsize)
{
  if (eff_bsize<=0) eff_bsize=bsize;

  MachLin::Forw(eff_bsize); //call to MachLin::Forw GEMM funtion

  // apply exp() on output and normalize
#ifdef BLAS_INTEL_MKL
  int s=eff_bsize*odim;
  VEXP(&s,data_out,data_out);
  REAL *optr=data_out;
  for (int b=0; b<eff_bsize; b++) {
    REAL sum=0; // TODO: double
    for (int i=0; i<odim; i++) sum += *optr++;
    optr-=odim;
    sum = 1.0/sum;  // circumvent division in loop
    for (int i=0; i<odim; i++) *optr++ *= sum;
  }
#else

  GpuMachSoftmaxForw(bsize,odim,data_out);

#endif
}

void MachSoftmax::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
    // derivate softmax activation function
    //   do_i / da_k = o_i (kronecker_ik - o_k)
    // we suppose that do_i/da_k vanishes in the error function !!
    //             = o_i (1 - o_i)
//    printf("entering MachSoftmax Backw.... \n");
#if 0
   // this can't be done here since the result depends
   // on the error function (we must derivate each output w/r
   // to ALL other outputs. This can't be stored in one vector)
   //   dE/da_i = sum_k dE/do_k do_k/da_i
   // On the other hand, many terms vanish with usual error functions

  REAL *aptr = data_out;
  REAL *gptr = grad_out;
```

```
  if (eff_bsize<=0) eff_bsize=bsize;
  if (!grad_out)
    Error("MachSoftmax::Backw(): output gradient is not set");

  for (int b=0; b<eff_bsize; b++) {
    REAL o;
    for (int i=0; i<odim; i++) {
      o=*optr++;
      *gptr++ *= o * (1-o);
    }
  }
#endif

  MachLin::Backw(lrate, wdecay, eff_bsize);
}
```

D.42 MachSoftmax.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachSoftmax.h,v 1.8 2010/01/25 12:27:07 schwenk Exp $
 *
 * softmax machine:  a_i = exp(a_i) / sum_k a_k
 * with a_k is the kth output of a linear machine
 */

#ifndef _MachSoftmax_h
#define _MachSoftmax_h

#include "MachLin.h"

class MachSoftmax : public MachLin
```

```
{
public:
  MachSoftmax(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~MachSoftmax();
  virtual int GetMType() {return file_header_mtype_softmax;};     //
get type of machine
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
};

#endif
```

D.43 MachTab.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTab.cpp,v 1.14 2010/01/26 19:37:22 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <stdlib.h>
extern double drand48();

#include "Tools.h"
#include "MachTab.h"
#include "cuda_runtime.h"
#include "Gpu.cuh"
#include "npps.h"
```

```
void MachTab::do_alloc()
{
  if (!ext_alloc) {
    //t = new REAL[idim*odim];
    cudaMallocManaged(&t, idim*odim*sizeof(REAL));
    if (!t) Error ("can't allocate memory for table look-up machine");
  }
  else
    ;
}


MachTab::MachTab(const int p_idim, const int p_odim, const int p_bsize,
const int p_nbfw, const int p_nbbw)
 : Mach(1, p_odim, p_bsize, p_nbfw, p_nbbw), ext_alloc(false)
{
  if (p_idim<=0) Error("Table machine: illegal value of input
dimension");
  if (p_odim<=0) Error("Table machine: illegal value of output
dimension");
  idim = p_idim; // override 1 in call to Mach()

  do_alloc();
}

MachTab::MachTab(REAL *ext_table,
      const int p_idim, const int p_odim, const int p_bsize,
      const int p_nbfw, const int p_nbbw)
 : Mach(1, p_odim, p_bsize, p_nbfw, p_nbbw), ext_alloc(true)
{
  if (p_idim<0) Error("Table machine: illegal value of input
dimension");
  if (p_odim<0) Error("Table machine: illegal value of output
dimension");
  idim = p_idim; // override 1 in call to Mach()

  //if (!ext_table) Error ("Table look-up machine: provided address is
NULL");
  t=ext_table;
  do_alloc();
}

MachTab::~MachTab()
{
  //if (!ext_alloc & (t!=NULL)) delete [] t;
  cudaFree(t);
}

void MachTab::TableConst(const REAL val)
{
  for (int i=0; i<idim*odim; i++) t[i]=val;
}

void MachTab::TableRandom(const REAL range)
{
  REAL c=range*2.0;
```

```
  for (int i=0; i<idim*odim; i++) t[i]=c*(drand48()-0.5);
}

void MachTab::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on table look-up machine" << endl;
    Mach::Info(detailed,txt);
  }
  else {
    printf("%sMachTab 1[%d]-%d, bs=%d, passes=%d/%d\n", txt, idim,
odim, bsize, nb_forw, nb_backw);
  }
}

//-----------------------------------------------
// File output
//-----------------------------------------------

void MachTab::WriteParams(ofstream &of)
{

  Mach::WriteParams(of);
  of.write((char*) &ext_alloc, sizeof(int));
}


void MachTab::WriteData(ofstream &outf) {
  int i=0, s=sizeof(REAL);
  if (ext_alloc) {
    outf.write((char*) &i, sizeof(int));
    outf.write((char*) &s, sizeof(int));
  }
  else {
    i=idim*odim;
    outf.write((char*) &i, sizeof(int));
    outf.write((char*) &s, sizeof(int));
    outf.write((char*) t,odim*idim*sizeof(REAL));
  }
}

//-----------------------------------------------
// File input
//-----------------------------------------------

void MachTab::ReadParams(ifstream &inpf, bool with_alloc)
{

  Mach::ReadParams(inpf, false);
  inpf.read((char*) &ext_alloc, sizeof(int));
  do_alloc();
}

void MachTab::ReadData(ifstream &inpf, ptrdiff_t s)
{
  ptrdiff_t se=odim*idim;
```

```
  if (ext_alloc) {
    if (s>0) {
      fprintf(stderr,"internal error in file, table look-up machine has
external allocation, but %u elements of data are provided\n",(uint)s);
      Error();
    }
    return; // address will be filled in by MachPar
  }
  else if (s!=se) {
    fprintf(stderr,"data block of table look-up machine has %u elements
- %u were expected)",(uint) s, (uint) se);
    Error();
  }
  Mach::ReadData(inpf, 0);
  inpf.read((char*) t,odim*idim*sizeof(REAL));
}


//-----------------------------------------------
// Training
//-----------------------------------------------

void MachTab::Forw(int eff_bsize)
{
  if (!data_in)
    Error("MachTab::Forw(): input data is not set");

  if (eff_bsize<=0) eff_bsize=bsize;
  cudaDeviceSynchronize();
  GpuMachTabForw(eff_bsize, odim, data_in, t, data_out);
//  printf("MachTab kernel return... \n");
//  REAL *optr=data_out;
////  printf("MachPar Tab Forw pointers ok... \n");
//  for (int b=0; b<eff_bsize; b++) {
//    int idx= (int) data_in[b];
//    if (idx<0 || idx>=idim) {
//      fprintf(stderr,"ERROR: illegal index (%d) in table look-up
machine, should be in [0,%d[\n", idx, idim);
//      Error();
//    }
//    memcpy(optr, t+idx*odim, odim*sizeof(REAL));
//    optr+=odim;
//  }
  nb_forw+=eff_bsize;
//  printf("MachTab forw ok!!!! \n");
}


void MachTab::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
  // table[wid] = table[wid] + lrate * grad_out[wid] * data_in[wid]
      cudaDeviceSynchronize();
  REAL *gptr = grad_out;
  for (int b=0; b<eff_bsize; b++) {
    int idx= (int) data_in[b];
    if (idx<0 || idx>=idim) {
```

```
      fprintf(stderr,"ERROR: illegal index (%d) in table look-up
machine (backw), should be in [0,%d[\n", idx, idim);
      Error();
    }
    REAL *tptr=t+idx*odim;
#ifdef DEBUG
    printf("  B %d idx=%d\n",b,idx);
    printf("  grad:"); for (int i=idx-2;i<=idx+2;i++) printf("
%f",gptr[i]); printf("\n");
    printf("   tab:");  for (int i=-2;i<=2;i++) printf(" %f",tptr[i]);
printf("\n");
#endif
    for (int i=0; i<odim; i++) *tptr++ += lrate * *gptr++;
    grad_in[b]=0; // we don't backprop to the input of a tbale look-up
machine
  }
    //          printf("MachTab Backw kernel call... \n");
//              GpuMachTabBackw(lrate,eff_bsize, odim, data_in, t,
grad_out);
          // we don't backprop to the input of a table look-up machine
//              nppsSet_32f(0.0,grad_in,eff_bsize);
    //          printf("MachTab Backw kernel call complete!! \n");
//              cudaDeviceSynchronize();
}
```

D.44 MachTab.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTab.h,v 1.9 2010/01/25 12:27:07 schwenk Exp $
 *
 * Table lookup machine:
 *   - input = index in table
```

```
 *   - output = ith line of table
 */

#ifndef _MachTab_h
#define _MachTab_h

#include "Mach.h"

class MachTab : public Mach
{
private:
  bool ext_alloc; // flag to indicate whether table was allocated
intertnally
  virtual void do_alloc();    // perform allocation of dynamic data
structures
protected:
  REAL *t;        // look-up table
  virtual void WriteParams(ofstream&);
  virtual void ReadParams(ifstream&, bool =true);
  virtual void ReadData(ifstream&, ptrdiff_t); // read binary data
  virtual void WriteData(ofstream&); // write binary data
  virtual int GetIdim() {return 1; } // we use idim internally as the
dim of the table entries
public:
  MachTab(const int=1, const int=1, const int=1, const int=0, const
int=0); // TODO: idim,odim init ??
  MachTab(REAL*, const int, const int, const int=1, const int=0, const
int=0);
  virtual ~MachTab();
  virtual int GetMType() {return file_header_mtype_tab;};   // get type
of machine
  virtual void TableConst(const REAL val);      // init table with
constant values
  virtual void TableRandom(const REAL range);   // random init of table
in [-range, range]
  virtual REAL *GetTabAdr() {return t; }  //
  virtual void SetTabAdr(REAL *p_adr) {t=p_adr; }     //
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
};

#endif
```

D.45 MachTanh.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
```

```
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTanh.cpp,v 1.14 2010/01/26 11:05:27 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <math.h>
//extern double drand48();
#include "my_cuda.h"
//#include "Tools.h"
#include "MachTanh.h"
#include "Blas.h"
#include "cuda_runtime.h"


MachTanh::MachTanh(const int p_idim, const int p_odim, const int
p_bsize, const int p_nbfw, const int p_nbbw)
 : MachLin(p_idim, p_odim, p_bsize, p_nbfw, p_nbbw)
{
      cudaMalloc((void **)&tmp_tanh, odim*bsize * sizeof(REAL));
}

MachTanh::~MachTanh()
{
      cudaFree(tmp_tanh);
}


//------------------------------------------------
// Tools
//------------------------------------------------

void MachTanh::Info(bool detailed, char *txt)
{
  if (detailed) {
    cout << "Information on tanh machine" << endl;
    MachLin::Info(detailed,txt);
  }
  else {
    printf("%sMachTanh %d-%d, bs=%d, passes=%d/%d\n", txt, idim, odim,
bsize, nb_forw, nb_backw);
```

```
  }
}

//-------------------------------------------------
// Training
//-------------------------------------------------

void MachTanh::Forw(int eff_bsize)
{

      float alpha = 1.0f;
      float beta = 1.0f;

      if (eff_bsize<=0) eff_bsize=bsize;
      MachLin::Forw(eff_bsize);

    // apply tanh() on output
#ifdef BLAS_INTEL_MKL
  int s=eff_bsize*odim;
  VTANH(&s,data_out,data_out);
#else
//  for (int i=0; i<eff_bsize*odim; i++) data_out[i]=tanh(data_out[i]);

  int s=eff_bsize*odim;
  nppsMulC_32f_I(2.0,data_out,s);         // 2*x
  nppsExp_32f_I(data_out,s);              // exp(2*x)
  nppsAddC_32f(data_out,1.0,tmp_tanh,s);  // tmp=exp(2*x)+1
  nppsSubC_32f_I(1.0,data_out,s);         // exp(2*x)-1
  nppsDiv_32f_I(tmp_tanh,data_out,s);           // (exp(2*x)-1) /
(exp(2*x)+1)

#endif
}

void MachTanh::Backw(const float lrate, const float wdecay, int
eff_bsize)
{
    // derivate tanh activation function
    // multiply grad_hidden by derivatives of hidden layer activities
(tanh)
    // grad_out = grad_out .* f'(data_out)
    //          = grad_out .* ( 1 - data_out^2 )

  REAL *aptr = data_out;
  REAL *gptr = grad_out;

  if (eff_bsize<=0) eff_bsize=bsize;
//  if (!grad_out)
//    Error("MachTanh::Backw(): output gradient is not set");
//  for (int i=0; i<odim*eff_bsize; i++) {
//    REAL val = *aptr++;
//    *gptr++ *= (1.0 - val * val);
//  }

      int d=odim*eff_bsize;
      nppsSqr_32f_I(data_out,d);
      nppsSubCRev_32f_I(1.0,data_out,d);
```

```
    nppsMul_32f_I(data_out,grad_out,d);

      MachLin::Backw(lrate, wdecay, eff_bsize);

}
```

D.46 MachTanh.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: MachTanh.h,v 1.7 2010/01/25 12:27:07 schwenk Exp $
 *
 * sigmoidal machine:  output = tanh(weights * input + biases)
 */

#ifndef _MachTanh_h
#define _MachTanh_h

#include "MachLin.h"

class MachTanh : public MachLin
{
protected:
      REAL *tmp_tanh;
public:
  MachTanh(const int=0, const int=0, const int=1, const int=0, const
int=0);
  virtual ~MachTanh();
  virtual int GetMType() {return file_header_mtype_tanh;};  // get type
of machine
  virtual void Info(bool=false, char *txt=(char*)""); // display
(detailed) information on machine
  virtual void Forw(int=0);   // calculate outputs for current inputs
```

```
    // backprop gradients from output to input and update all weights
  virtual void Backw (const float lrate, const float wdecay, int=0);
};

#endif
```

D.47 Nbest.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NBest.h,v 1.6 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _NBEST_H_
#define _NBEST_H_

using namespace std;

#include <iostream>
#include <fstream>
#include <string>
#include <vector>

#include "Toolsgz.h"
#include "Hypo.h"
#include "NbestLM.h"

class NBest {
  int               id;
  string           src;
  vector<Hypo> nbest;
  bool ParseLine(inputfilestream &inpf, const int n);
 public:
```

```
  NBest(inputfilestream&, const int=0);
  ~NBest();
  int NbNBest() {return nbest.size(); };
  void CalcGlobal(Weights&);
  void Sort(); // largest values first
  void Write(outputfilestream&, int=0);
  void AddID(const int offs);
  void RescoreLM(NbestLM&, const int);
};


#endif
```

D.48 NbestLM.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NbestLM.cpp,v 1.6 2010/01/26 19:37:22 schwenk Exp $
 */

#include <iostream>
#include <stdlib.h>     // exit()
#include "NbestLM.h"

bool NbestLM::Read (const string &fname, int const order)
{
  cerr << "Read() of virtual class NbestLM called" << endl;
  exit(1);
  return false;
}
```

D.49 NbestLM.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: NbestLM.h,v 1.6 2010/01/26 19:37:22 schwenk Exp $
 */

#ifndef _NBESTLM_H_
#define _NBESTLM_H_

using namespace std;

#include <string>
#include <vector>
#include "Hypo.h"

#define RESCORE_MODE_BOS 1
#define RESCORE_MODE_EOS 2

class NbestLM {
protected:
  string fname;  // translation
  int lm_order;  // order of NbestLM
  int mode;
  vector<int> nb_ngrams;  // nb of ngrams per order, nb_ngrams[0] is
voc. size
public:
  NbestLM() : mode(RESCORE_MODE_BOS | RESCORE_MODE_EOS) {};
  virtual ~NbestLM() {};
  virtual float GetValue() {return 0; };
  virtual bool Read (const string &, int const order = 4);
  virtual void RescoreHyp (Hypo &hyp, const int lm_pos) {};   // recalc
LM score on hypothesis
```

```
  virtual void FinishPending() {};  // finish pending requests, only
used for CSLM
};

#endif
```

D.50 Tools.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Tools.cpp,v 1.3 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>

#include "Tools.h"

void Error(void)
{
  exit(1);
}

void Error(const char *txt)
{
  cerr << "ERROR: " << txt << endl;
  exit(1);
}

int ReadInt(ifstream &inpf, const string &name, int minval,int maxval)
{
  string buf;
  inpf >> buf;
```

```
  if (buf!=name) {
    cerr << "FileRead: found field '" << buf << "' while looking for '"
<< name << "'";
    Error("");
  }

  int val;
  inpf >> val;
  if (val<minval || val>maxval) {
    cerr << "FileRead: values for " << name << "must be in
["<<minval<<","<<maxval<<"]";
    Error("");
  }

  return val;
}
```

D.51 Tools.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Tools.h,v 1.15 2010/01/25 12:53:31 schwenk Exp $
 */

#ifndef _Tools_h
#define _Tools_h

#include <iostream>
#include <fstream>
#include <string.h>     // memcpy()
#include <stdlib.h>     // exit()
#include <math.h>
```

```
typedef float REAL;
typedef unsigned int uint;
typedef long unsigned int luint;

//
// general purpose helper functions
//
#ifdef DEBUG
# define TRACE(txt) cout << txt;
# define debug(F) printf(F)
# define debug1(F,a) printf(F,a)
# define debug2(F,a,b) printf(F,a,b)
# define debug3(F,a,b,c) printf(F,a,b,c)
# define debug4(F,a,b,c,d) printf(F,a,b,c,d)
# define debug5(F,a,b,c,d,e) printf(F,a,b,c,d,e)
# define debug6(F,a,b,c,d,e,f) printf(F,a,b,c,d,e,f)
# define debug7(F,a,b,c,d,e,f,h) printf(F,a,b,c,d,e,f,h)
# define debug8(F,a,b,c,d,e,f,h,i) printf(F,a,b,c,d,e,f,h,i)
#else
# define TRACE(txt)
# define debug(F)
# define debug1(F,a)
# define debug2(F,a,b)
# define debug3(F,a,b,c)
# define debug4(F,a,b,c,d)
# define debug5(F,a,b,c,d,e)
# define debug6(F,a,b,c,d,e,f)
# define debug7(F,a,b,c,d,e,f,h)
# define debug8(F,a,b,c,d,e,f,h,i)
#endif

void Error(void);
void Error(const char *txt);

#define CHECK_FILE(ifs,fname) if(!ifs) { perror(fname); Error(); }

//
// parsing of ASCII files
//
int ReadInt(ifstream&,const string&,int=0,int=2147483647); // TODO:
MAXINT
float ReadFloat(ifstream&,const string&,float=0,float=3.4e38); // TODO:
MAXFLOAT
string ReadText(ifstream&,const string&);

#endif
```

D.52 Toolsgz.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
```

```
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Toolsgz.h,v 1.4 2010/01/26 19:43:09 schwenk Exp $
 */

#ifndef TOOLSGZ_H
#define TOOLSGZ_H

using namespace std;

#include <stdexcept>
#include <limits>
#include <cstring>      // for memmove
#include <zlib.h>
#include <vector>
#include <map>
#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <streambuf>

#define US_NOSET (numeric_limits<unsigned short>::max())
#define MAX_LINE  1024


//
//
//

class Weights {
  vector<float> val;
 public:
  Weights() {};
  ~Weights() {};
  int Read(const char *);
  friend class Hypo;
};


class gzfilebuf : public std::streambuf {
```

```
public:
  gzfilebuf(const char *filename)
  { _gzf = gzopen(filename, "rb");
    setg (_buff+sizeof(int),     // beginning of putback area
          _buff+sizeof(int),     // read position
          _buff+sizeof(int));    // end position
  }
  gzfilebuf(const char *filename, int dummy)
  { _gzf = gzopen(filename, "w+b");
    setg (_buff+sizeof(int),     // beginning of putback area
          _buff+sizeof(int),     // read position
          _buff+sizeof(int));    // end position
  }
  ~gzfilebuf() { gzclose(_gzf); }
protected:
  virtual int_type overflow (int_type c) {
      throw;
  }

  // write multiple characters
  virtual
  std::streamsize xsputn (const char* s,
                          std::streamsize num) {
      throw;
  }

  virtual std::streampos seekpos ( std::streampos sp,
std::ios_base::openmode which = std::ios_base::in | std::ios_base::out
){ throw;
  }

  //read one character
  virtual int_type underflow () {
    // is read position before end of _buff?
        if (gptr() < egptr()) {
            return traits_type::to_int_type(*gptr());
        }

        /* process size of putback area
         * - use number of characters read
         * - but at most four
         */
        unsigned int numPutback = gptr() - eback();
        if (numPutback > sizeof(int)) {
            numPutback = sizeof(int);
        }

        /* copy up to four characters previously read into
         * the putback _buff (area of first four characters)
         */
        memmove (_buff+(sizeof(int)-numPutback), gptr()-numPutback,
                      numPutback);

        // read new characters
        int num = gzread(_gzf, _buff+sizeof(int), _buffsize-
sizeof(int));
        if (num <= 0) {
```

```
            // ERROR or EOF
            return EOF;
        }

        // reset _buff pointers
        setg (_buff+(sizeof(int)-numPutback),   // beginning of putback
area
              _buff+sizeof(int),                // read position
              _buff+sizeof(int)+num);           // end of buffer

        // return next character
        return traits_type::to_int_type(*gptr());
  }

  std::streamsize xsgetn (char* s,
                          std::streamsize num) {
    return gzread(_gzf,s,num);
  }

private:
  gzFile _gzf;
  static const unsigned int _buffsize = 1024;
  char _buff[_buffsize];
};

//
//
//

class inputfilestream : public std::istream
{
protected:
        std::streambuf *m_streambuf;
      bool _good;
public:

        inputfilestream(const std::string &filePath);
        ~inputfilestream();
      bool good(){return _good;}
        void close();
};

class outputfilestream : public std::ostream
{
protected:
        std::streambuf *m_streambuf;
      bool _good;
public:

        outputfilestream(const std::string &filePath);
        ~outputfilestream();
      bool good(){return _good;}
        void close();
};

/***************************************************
 *
```

```
 * Compressed File IO
 */

class gzifstream : public std::istream
{
protected:
  std::streambuf *gz_streambuf;
  bool _fail;
public:
  gzifstream() : gz_streambuf(0), _fail(true) {};
  ~gzifstream() {if (gz_streambuf) delete(gz_streambuf); };
  void open(char*);
  bool fail() {return _fail;}
  void close() {};
};


class gzofstream : public std::ostream
{
protected:
  std::streambuf *gz_streambuf;
  bool _fail;
public:
  gzofstream() : gz_streambuf(0), _fail(true) {};
  ~gzofstream() {if (gz_streambuf) delete(gz_streambuf); };
  void open(char*);
  bool fail() {return _fail;}
  void close() {};
};


//*****************************************************


template<typename T>
inline T Scan(const std::string &input)
{
         std::stringstream stream(input);
         T ret;
         stream >> ret;
         return ret;
}


#endif
```

D.53 Trainer.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
```

```
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Trainer.cpp,v 1.9 2010/01/25 12:27:07 schwenk Exp $
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>

#include "Tools.h"
#include "Mach.h"
#include "Trainer.h"
#include "cuda_runtime.h"

Trainer::Trainer (Mach *pmach, ErrFct *perrfct,
              char *train_fname, char *dev_fname,
              REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,
              int p_maxep, int p_ep)
 : mach(pmach), errfct(perrfct),
   lrate_beg(p_lr_beg), lrate_mult(p_lr_mult), wdecay(p_wd),
   nb_epoch(p_ep), max_epoch(p_maxep)
{
  char      msg[1024];

  idim=mach->GetIdim(); odim=mach->GetOdim(); bsize=mach->GetBsize();
  if (train_fname) {
    data_train = new Data(train_fname);

    if (idim != data_train->GetIdim()) {
      sprintf(msg,"Trainer: input dimension of the training data (%d)
does not match the one of the machine (%d)\n", data_train->GetIdim(),
idim);
      Error(msg);
    }
    if (odim != data_train->GetOdim()) {
      sprintf(msg,"Trainer: ouput dimension of the training data (%d)
does not match the one of the machine (%d)\n", data_train->GetOdim(),
odim);
      Error(msg);
    }
  }
```

```
  else
    data_train=NULL;

  if (dev_fname) {
    data_dev = new Data(dev_fname);
    if (idim != data_dev->GetIdim())
      Error("Trainer: input dimension of the validation data does not
match the one of the machine\n");
    if (odim != data_dev->GetOdim())
      Error("Trainer: output dimension of the validation data does not
match the one of the machine\n");
    }
  else
    data_dev=NULL;

  //buf_input = new REAL[idim*bsize];
  cudaMallocManaged(&buf_input, idim*bsize*sizeof(REAL));
  //buf_target = new REAL[odim*bsize];
  cudaMallocManaged(&buf_target, odim*bsize*sizeof(REAL));
  // memory for the output gradient is allocated by the error function
}

//*********************************************************************
*****************

Trainer::~Trainer ()
{
  //if (data_train) delete data_train;
  //if (data_dev) delete data_dev;
  //delete [] buf_input;
  //delete [] buf_target;
  cudaFree(buf_input);
  cudaFree(buf_target);
}

//*********************************************************************
*****************
// default lrate = mach->lrate_begin / (1.0 + total_n_ex_seen * mach-
>lrate_mult);
// default wdecay: constant

void Trainer::SetLrate()
{
  lrate = lrate_beg / (1.0 + mach->GetNbForw() * lrate_mult);
}

//*********************************************************************
*****************

REAL Trainer::Train()
{
//    printf("entering Trainer::Train.... \n");
#ifdef DEBUG
  printf("*****************\n");
  printf("Trainer::Train():\n");
  printf(" -  data_in: %p \n", (void*) buf_input);
  printf(" -   target: %p \n", (void*) buf_target);
```

```
  printf(" - grad_out: %p \n", (void*) errfct->GetGrad());
#endif
  data_train->Rewind();

  REAL err=0;
  nb_ex=0;
  mach->SetDataIn(buf_input);
  mach->SetGradOut(errfct->GetGrad());
  errfct->SetOutput(mach->GetDataOut());
  errfct->SetTarget(buf_target);
  bool data_available;
  do {
      // get a bunch of data
    int n=0;
    data_available = true;
    while (n < mach->GetBsize() && data_available)
    {
      data_available = data_train->Next();
      if (!data_available) break;
      memcpy(buf_input  + n*idim, data_train->input,
idim*sizeof(REAL));
      memcpy(buf_target + n*odim, data_train->target,
odim*sizeof(REAL));
      n++;
    }

    if (n>0)
    {
      mach->Forw(n);
      err += errfct->CalcGrad(n);
#ifdef DEBUG
      printf("OUTPUT:") ; for (int i=0;i<odim; i++) printf("
%4.1f",mach->GetDataOut()[i]); printf("\n");
      printf("TARGET:") ; for (int i=0;i<odim; i++) printf("
%4.1f",data_train->target[i]); printf("\n");
      printf("  GRAD:") ; for (int i=0;i<odim; i++) printf("
%4.1f",errfct->GetGrad()[i]); printf("\n");
#endif
      SetLrate();
      mach->Backw(lrate, wdecay, n);
    }

    nb_ex += n;
  } while (data_available);
  err /= nb_ex;

  return err;
}

//*******************************************************************
*****************
// This should be overriden to do a task-specific validation

REAL Trainer::TestDev()
{
  if (!data_dev) return -1;
```

```
  int nb_ex_dev=0;
  REAL err=0;
  data_dev->Rewind();
  mach->SetDataIn(buf_input);
  errfct->SetOutput(mach->GetDataOut());
  errfct->SetTarget(buf_target);
  bool data_available;
  do {
      // get a bunch of data
    int n=0;
    data_available = true;
    while (n < mach->GetBsize() && data_available) {
      data_available = data_dev->Next();
      if (!data_available) break;
      memcpy(buf_input  + n*idim, data_dev->input, idim*sizeof(REAL));
      memcpy(buf_target + n*odim, data_dev->target, odim*sizeof(REAL));
      n++;
    }

      // process the bunch
    if (n>0) {
      mach->Forw(n);
      err += errfct->CalcValue(n);
#ifdef DEBUG
      printf(" INPUT:") ; for (int i=0;i<idim; i++) printf("
%4.1f",mach->GetDataIn()[i]); printf("\n");
      printf("OUTPUT:") ; for (int i=0;i<odim; i++) printf("
%4.1f",mach->GetDataOut()[i]); printf("\n");
      printf("TARGET:") ; for (int i=0;i<odim; i++) printf("
%4.1f",data_dev->target[i]); printf(" -> %f\n",errfct->CalcValue(n));
#endif
    }

    nb_ex_dev += n;
  } while (data_available);

  if (nb_ex_dev>0) return err/nb_ex_dev;
  return -1;
}


//*****************************************************************
*************
// simple training routine

void Trainer::TrainAndTest ()
{
  const int hlen=256;
  char hostname[hlen];
  gethostname(hostname, hlen); hostname[hlen-1]=0;
  cout << "Starting training on host " << hostname << " pid " <<
getpid() << endl;
  cout << " - training on " << data_train->GetFname() << endl;
  if (data_dev)
    cout << " - validation on " << data_dev->GetFname() << endl;
  cout << " - stopping training at " << max_epoch << " epochs" << endl;
  mach->Info();
```

```
  while (!Converged()) {
    InfoPre();
    err_train = Train();
    InfoPost();

    cout << " - starting validation ..."; cout.flush();
    err_dev = TestDev();
    if (err_dev<0)
      cout << " avrg error: no examples !?" << endl;
    else
      cout << " avrg error: " << err_dev << endl;
  }
  cout << "Training stopped" << endl;
  mach->Info();
  //mach->Write();
}

//*************************************************************************
*****************

bool Trainer::Converged ()
{
  return (nb_epoch >= max_epoch);
}

//*************************************************************************
*****************
// information before starting an epoch

void Trainer::InfoPre ()
{
  time_t now;
  time(&now); // TODO: ctime is not rentrant ! use ctime_r() instead if
needed
  cout << "Starting epoch " << ++nb_epoch << " at " << ctime(&now);

  SetLrate();
  fprintf(stdout, " - intial lrate=%6.4e, wdecay=%6.4e\n", lrate,
wdecay);
}

//*************************************************************************
*****************
// information after finishing an epoch

void Trainer::InfoPost ()
{
  cout << " - epoch finished, " << nb_ex << " examples seen, average
error: " << err_train << endl;
}
```

D.54 Trainer.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: Trainer.h,v 1.5 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _Trainer_h
#define _Trainer_h

#include <iostream>
#include "Tools.h"
#include "Mach.h"
#include "ErrFct.h"
#include "Data.h"


class Trainer
{
private:
protected:
  Mach  *mach;                // network to train
  ErrFct *errfct;       // error fucntion to use
  Data  *data_train;          // training data to use
  Data  *data_dev;            // development data to use
    // buffer to store bsize examples
  REAL  *buf_input;
  REAL  *buf_target;
    // current learning rates
  REAL lrate_beg, lrate_mult; // params for exponential decay
  REAL lrate, wdecay;         // current values
    // stats
  int nb_ex;                  // during one epoch
  int nb_epoch;               // total nb of epochs
```

```
  int max_epoch;        // max numebr of epochs
  int idim, odim, bsize;      // copied here for faster access
  REAL err_train;       // average error during training
  REAL err_dev;               // average error during testing
   // internal helper functions
  virtual void SetLrate();          // modify learning rates
  virtual bool Converged();         // return TRUE if training has
converged or should be stopped
  virtual void InfoPre();           // dump information before starting
a new training epoch
  virtual void InfoPost();          // dump information after finishing
a training epoch
public:
  Trainer(Mach*, ErrFct*, char*, char* =NULL,   // mach, errfct, train,
dev
          float = 0.01, float =0, float =0,     // lrate_beg,
lrate_mult, wdecay
        int =10, int =0);                 // max epochs, current epoch
  virtual ~Trainer();
  virtual REAL Train();             // train for one epoch
  virtual REAL TestDev();           // test current network on dev data
                                // returns obtained error (-1 if error)
  virtual void TrainAndTest();            // main training routine for
X iterations
};

#endif
```

D.55 TrainerNgram.cpp

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: TrainerNgram.cpp,v 1.13 2010/01/25 12:27:07 schwenk Exp $
```

```
 */

using namespace std;
#include <iostream>
#include <unistd.h>
#include <time.h>
#include "my_cuda.h"
#include "Tools.h"
#include "Mach.h"
#include "TrainerNgram.h"
#include "cuda_runtime.h"

//cudnnHandle_t cudnn_handle;
//cudnnStatus_t cudnn_stat;
cublasHandle_t cublas_handle;
cublasStatus_t cublas_stat;

TrainerNgram::TrainerNgram (Mach *pmach, ErrFct *perrfct,
      char *train_fname, char *dev_fname,
      REAL p_lr_beg, REAL p_lr_mult, REAL p_wd,
      int p_maxep, int p_ep)
 :
Trainer(pmach,perrfct,NULL,NULL,p_lr_beg,p_lr_mult,p_wd,p_maxep,p_ep),
   order(0)
{
  char      msg[1024];

  idim=mach->GetIdim(); odim=mach->GetOdim(); bsize=mach->GetBsize();

  if (odim < 16) {
    sprintf(msg,"TrainerNgram: output dimension of the machine is
suspiciously small (%d)\n", odim);
    Error(msg);
  }

  if (train_fname) {
    data_train = new Data(train_fname);
    if (idim != data_train->GetIdim()) {
      sprintf(msg,"TrainerNgram: input dimension of the training data
(%d) does not match the one of the machine (%d)\n", data_train-
>GetIdim(), idim);
      Error(msg);
    }
    if (data_train->GetOdim() != 1) {
      sprintf(msg,"TrainerNgram: output dimension of the training data
should be 1, found %d\n", data_train->GetOdim());
      Error(msg);
    }
  }
  else
    data_train=NULL;

  if (dev_fname) {
    data_dev = new Data(dev_fname);
    if (idim != data_dev->GetIdim()) {
```

```
      sprintf(msg,"TrainerNgram: input dimension of the validation data
(%d) does not match the one of the machine (%d)\n", data_dev-
>GetIdim(), idim);
      Error(msg);
    }
    if (data_dev->GetOdim() != 1) {
      sprintf(msg,"TrainerNgram: output dimension of the validation
data should be 1, found %d\n", data_dev->GetOdim());
      Error(msg);
    }
  }
  else
    data_dev=NULL;
}

TrainerNgram::TrainerNgram (Mach *pmach, ErrFct *perrfct, Data &data)
 : Trainer(pmach,perrfct,NULL,NULL,0,0,0,0,0),
    order(0)
{
  char      msg[1024];

  idim=mach->GetIdim(); odim=mach->GetOdim(); bsize=mach->GetBsize();

  if (odim < 16) {
    sprintf(msg,"TrainerNgram: output dimension of the machine is
suspiciously small (%d)\n", odim);
    Error(msg);
  }

  data_train=NULL;
  data_dev=&data;

  if (idim != data_dev->GetIdim()) {
    sprintf(msg,"TrainerNgram: input dimension of the validation data
(%d) does not match the one of the machine (%d)\n", data_dev-
>GetIdim(), idim);
    Error(msg);
  }
  if (data_dev->GetOdim() != 1) {
    sprintf(msg,"TrainerNgram: output dimension of the validation data
should be 1, found %d\n", data_dev->GetOdim());
    Error(msg);
  }
}

//*********************************************************************
****************
// default lrate = mach->lrate_begin / (1.0 + total_n_ex_seen * mach-
>lrate_mult);
// default wdecay: constant

void TrainerNgram::SetLrate()
{
  lrate = lrate_beg / (1.0 + mach->GetNbForw() * lrate_mult);
}
```

```
//*********************************************************************
****************

REAL TrainerNgram::Train()
{

  if (!data_train) return -1;
#ifdef DEBUG
  printf("*****************\n");
  printf("TrainerNgram::Train():\n");
  printf(" -  data_in: %p \n", (void*) buf_input);
  printf(" -   target: %p \n", (void*) buf_target);
  printf(" - grad_out: %p \n", (void*) errfct->GetGrad());
#endif
  data_train->Rewind();

  REAL log_sum=0;
  nb_ex=0;
  mach->SetDataIn(buf_input);
  mach->SetGradOut(errfct->GetGrad());
  errfct->SetOutput(mach->GetDataOut());
  errfct->SetTarget(buf_target);

  bool data_available;
  do
  {
      // get a bunch of data
      // TODO: exlude out of slist
    int n=0;
    data_available = true;
    while (n < mach->GetBsize() && data_available)
    {
      data_available = data_train->Next();
      if (!data_available) break;
      //printf("starting buf_input memcpy!!! \n");
      memcpy(buf_input  + n*idim, data_train->input,
idim*sizeof(REAL));
      //printf("buf_input memcpy ok, starting  buf_target memcpy!!!
\n");
      memcpy(buf_target + n*1, data_train->target, 1*sizeof(REAL));
      n++;
    }

    //if (nb_ex%1024==0) printf("."); fflush (stdout);
    if (n>0)
    {
      mach->Forw(n);
      log_sum += errfct->CalcGrad(n);
//      printf("CalcGrad complete!!! log_sum=%f \n",log_sum);
#ifdef DEBUG2
      int t=(int) data_train->target[0];
printf("OUTPUT:") ; for (int i=t-2;i<=t+2; i++) printf(" %f",mach-
>GetDataOut()[i]); printf("\n");
printf("TARGET:") ; for (int i=0;i<1; i++) printf(" %f",data_train-
>target[i]); printf("\n");
printf("  GRAD:") ; for (int i=t-2;i<=t+2; i++) printf(" %f",errfct-
>GetGrad()[i]); printf("\n");
```

```
#endif
      SetLrate();
      mach->Backw(lrate, wdecay, n);
    }

    nb_ex += n;
  } while (data_available);
  cudaDeviceSynchronize();
  if (nb_ex>0) return exp(-log_sum / (REAL) nb_ex);  // return
perplexity

  return -1;
}

//**********************************************************************
*****************
// This should be overriden to do a task-specific validation

REAL TrainerNgram::TestDev(char *fname)
{
  if (!data_dev) return -1;

  if (fname) {
    Error("not yet implemented");
  }
//  cudaDeviceSynchronize();
  int nb_ex_dev=0;
  REAL log_sum=0;
  data_dev->Rewind();
  mach->SetDataIn(buf_input);
  errfct->SetOutput(mach->GetDataOut());
  errfct->SetTarget(buf_target);
  bool data_available;
  do {
      // get a bunch of data
      // TODO: exlude out of slist
    int n=0;
    data_available = true;
    while (n < mach->GetBsize() && data_available) {
      data_available = data_dev->Next();
      if (!data_available) break;
      memcpy(buf_input  + n*idim, data_dev->input, idim*sizeof(REAL));
      memcpy(buf_target + n*1, data_dev->target, 1*sizeof(REAL));
      n++;
    }

      // process the bunch
    if (n>0) {
#ifdef DEBUG
printf("in:"); for (int i=0;i<idim;i++) printf(" %f", buf_input[i]);
printf("-> trg:"); for (int i=0;i<1;i++) printf(" %f", buf_target[i]);
printf("\n");
#endif
      cudaDeviceSynchronize();
      mach->Forw(n);
      log_sum += errfct->CalcValue(n);
      if (fname) {
```

```
        Error();  // TODO: we should get access to the parts of bsize
      }
    }

    nb_ex_dev += n;
  } while (data_available);
  cudaDeviceSynchronize();
  if (nb_ex_dev>0) return exp(-log_sum / (REAL) nb_ex_dev);  // return
perplexity
  return -1;
}


//*********************************************************************
*************
// simple training routine

void TrainerNgram::TrainAndTest ()
{
  //cudaDeviceReset();

  if (!data_train) {
    cout << "No training data specified, training impossible" << endl;
    return;
  }

  cublas_stat = cublasCreate(&cublas_handle);
  if (cublas_stat != CUBLAS_STATUS_SUCCESS)
  {
        fprintf(stderr,"\n CuBLAS Initialization Failed \n");
        cudaDeviceReset();
        Error();
  };
//  cudnn_stat = cudnnCreate(&cudnn_handle);
//  if (cudnn_stat != CUDNN_STATUS_SUCCESS)
//  {
//      fprintf(stderr,"\n CuDNN Initialization Failed \n");
//      cudaDeviceReset();
//      Error();
//  };

  const int hlen=256;
  char hostname[hlen];
  gethostname(hostname, hlen); hostname[hlen-1]=0;
  cout << "Starting training on host " << hostname << " pid " <<
getpid() << endl;
  cout << " - training on " << data_train->GetFname() << endl;
  if (data_dev)
    cout << " - validation on " << data_dev->GetFname() << endl;
  cout << " - stopping training at " << max_epoch << " epochs" << endl;
  mach->Info();

  while (!Converged()) {
    InfoPre();
    err_train = Train();
    InfoPost();
```

```
    cout << " - starting validation ..."; cout.flush();
    err_dev = TestDev();
    if (err_dev<0)
      cout << " avrg error: no examples !?" << endl;
    else
      cout << " avrg error: " << err_dev << endl;
  }


  cout << "Training stopped" << endl;
  mach->Info();
  /* Shutdown */

//  cudnnDestroy(cudnn_handle);
  cublasDestroy(cublas_handle);
  //mach->Write();
}

//*********************************************************************
*****************

bool TrainerNgram::Converged ()
{
  return (nb_epoch >= max_epoch);
}

//*********************************************************************
*****************
// information before starting an epoch

void TrainerNgram::InfoPre ()
{
  time_t now;
  time(&now); // TODO: ctime is not rentrant ! use ctime_r() instead if
needed
  cout << "Starting epoch " << ++nb_epoch << " at " << ctime(&now);

  SetLrate();
  fprintf(stdout, " - intial lrate=%6.4e, wdecay=%6.4e\n", lrate,
wdecay);
}

//*********************************************************************
*****************
// information after finishing an epoch

void TrainerNgram::InfoPost ()
{
  cout << " - epoch finished, " << nb_ex << " examples seen, average
error: " << err_train << endl;
}
```

D.56 TrainerNgram.h

```
/*
 * This file is part of the continuous space language model toolkit for
large
 * vocabulary speech recognition and statistical machine translation.
 *
 * Copyright 2010, Holger Schwenk, LIUM, University of Le Mans, France
 *
 * The CSLM toolkit is free software; you can redistribute it and/or
modify it
 * under the terms of the GNU General Public License version 3 as
 * published by the Free Software Foundation
 *
 * This library is distributed in the hope that it will be useful, but
WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this library; if not, write to the Free Software
Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * $Id: TrainerNgram.h,v 1.5 2010/01/25 12:27:07 schwenk Exp $
 */

#ifndef _TrainerNgram_h
#define _TrainerNgram_h

#include <ostream>
#include "Tools.h"
#include "Mach.h"
#include "ErrFct.h"
#include "Data.h"
#include "Trainer.h"


class TrainerNgram : public Trainer
{
private:
    // copies of important fields
  int order;                  // from Data
protected:
   // internal helper functions
  virtual void SetLrate();          // modify learning rates
  virtual bool Converged();         // return TRUE if training has
converged or should be stopped
  virtual void InfoPre();           // dump information before starting
a new training epoch
  virtual void InfoPost();          // dump information after finishing
a training epoch
public:
```

```
  TrainerNgram(Mach*, ErrFct*, char*, char* =NULL,    // mach, errfct,
train, dev
          float = 0.01, float =0, float =0,     // lrate_beg,
lrate_mult, wdecay
        int =10, int =0);                 // max epochs, current epoch
  TrainerNgram(Mach*, ErrFct*, Data&);    // for testing only
  virtual REAL Train();             // train for one epoch
  virtual REAL TestDev(char * =NULL);     // test current network on
dev data
                              // returns obtained error (-1 if error)
  virtual void TrainAndTest();            // main training routine for
X iterations
};

#endif
```